Um esquema baseado em *blockchain* por *Proof-of-Download* para gerenciamento de direitos digitais e detecção de traidores

João Tito do Nascimento Silva[1], Felipe Z. da N. Costa[2], João Gondim[3]

[1]Departamento de Ciência da Computação - Universidade de Brasília (UnB)
Brasília-DF, Brasil

[2]CIn - Centro de Informática - Universidade Federal de Pernambuco

[3]Departamento de Engenharia Elétrica - Universidade de Brasília (UnB)
Brasília-DF, Brasil

{joao.tito@aluno.unb.br felipe@zimmerle.org, gondim@unb.br}

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) comprises 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.*

Resumo. *O licenciamento de ativos digitais é um aspecto importante dos mercados modernos, como o da música e do cinema. Neste contexto, a Gestão de Direitos Digitais (DRM) e a Detecção de Traidores (TD) compreendem um conjunto de ferramentas, técnicas e processos que tentam garantir o controle do uso de mídias licenciadas. Este trabalho contribui para esta área ao possibilitar que DRM e TD aconteçam em uma blockchain pública e altamente descentralizada sob um modelo de ameaça proposto, desenvolvendo novas técnicas criptográficas que, como mostram os resultados obtidos, são muito eficientes sem exigir hardware caro.*

1. Introdução

O licenciamento de ativos é parte importante de mercados como software, música e entretenimento, que são amplamente baseados em controle sobre mídia digital para monetização. O conjunto de ferramentas e técnicas que tenta garantir o licenciamento e controle sobre os ativos digitais é chamado de Gestão de Direitos Digitais (*Digital Rights Management* - DRM).

O objetivo de políticas de DRM é garantir que os ativos não percam seu valor ao impor um sistema de licenciamento durante o uso de um ativo. No entanto, estes não impedem completamente o compartilhamento ilegal de ativos. Nesse contexto, os mecanismos de Detecção de Traidores (*Traitor Detection* - TD) se tornam essenciais, permitindo rastrear a origem de uma cópia não licenciada de um ativo.

As principais contribuições deste trabalho para DRM e TD são um modelo de ameaça e técnicas criptográficas corrrespondentes que permitem que DRM e TD ocorram

em um ambiente público, transparente e altamente descentralizado baseado em *blockchain*, em contraste com pesquisa recente na área, que frequentemente se concentra em soluções centralizadas e baseadas em mecanismos não criptográficos, como apresentado na seção 2. Após apresentar trabalhos relacionados na área, os atores e o modelo de ameaça para DRM e TD são desenvolvidos na seção 3. Esse modelo é então utilizado como base para o desenvolvimento de um protocolo para *downloads* verificáveis dos conteúdos de ativos digitais, chamado *Provable Data Provisioning* (PDPr), na seção 4. Em seguida, o design da *blockchain* é especificado, começando com o manuseio de pagamentos na seção 5 e introduzindo aspectos de forja de blocos e mecanismos de consenso na seção 6. Os protocolos de PDPr são então incorporados na *blockchain* na seção 7. Finalmente, detalhes de implementação e resultados são apresentados na seção 8 e os objetivos para trabalhos futuros são discutidos na seção 9.

2. Trabalhos relacionados

Nesta seção, as contribuições de trabalhos recentes em DRM e TD e suas contribuições são discutidas e contrastadas com as contribuições deste trabalho.

Tanto [De León and Gupta 2017] quanto [Ciriello et al. 2023] fornecem ideias interessantes sobre a mecânica da interação entre DRM e modelos de negócios e propõem mecanismos que tentam atender a muitos aspectos diferentes desses modelos de negócios. No entanto, eles se concentram na indústria musical, tornando suas soluções mais difíceis de interoperar com outros sistemas e modelos. Além disso, os mecanismos propostos são baseados na gestão de endereços IP ou na existência de um gestor central dos ativos, criando a necessidade de confiança em uma terceira parte e dificultando a venda de ativos em aberto. Em contraste, aqui focamos no desenvolvimento de uma solução pública baseada em *blockchain* que permite a interoperabilidade com outros sistemas e modelos de negócios de forma mais flexível.

Em [Ma et al. 2018], o modelo e os mecanismos propostos foram elaborados com atenção a muitos aspectos importantes do DRM, porém, não é dada atenção suficiente aos pagamentos. Por outro lado, nossa proposta traz uma melhor harmonização entre pagamentos e *downloads* para um DRM mais robusto e uma melhor garantia para os clientes, ao interoperar com *smart contracts* em *blockchains* bem conhecidas e confiáveis.

Por fim, [Siddique and Fatima 2022] traz mecanismos interessantes para a unicidade dos ativos. No entanto, não é considerada a necessidade de manter os ativos confidenciais em um ambiente público. Em contraste, neste trabalho apresentam-se protocolos criptográficos e um esquema inspirado em *Proof-of-Download* [Costa et al. 2022] que permitem o fornecimento verificável dos ativos enquanto sua confidencialidade é mantida.

Portanto, embora existam propostas de soluções para DRM e TD com sistemas distribuídos, todas essas soluções carecem de protocolos verdadeiramente descentralizados capazes de conciliar a distribuição de ativos com aspectos financeiros. Por outro lado, ao utilizar técnicas criptográficas, este trabalho propõe um design de *blockchain* público e transparente com essas propriedades, sem comprometer a eficiência e o desempenho.

3. Atores e Modelo de Ameaça

3.1. Atores e requisitos

Neste trabalho, dois atores principais são considerados: o **provedor**, que tem interesse em fornecer acesso ao conteúdo de um ativo em troca de um pagamento; e o **cliente**, que está interessado em comprar direitos de acesso e uso de um ativo fornecido por algum provedor.

A interação entre eles ocorrerá em um ambiente público, onde terceiros poderão testemunhar o processo e verificar provas emitidas para confirmar a participação correta desses atores nos protocolos. Temos então duas categorias de verificadores: **verificadores de pagamento**, que verificam os pagamentos feitos pelos clientes com **provas de pagamento**; e **verificadores de *download***, que verificam se os provedores forneceram os ativos aos clientes com **provas de *download***.

O provedor também emitirá uma **licença** que é *irrevogável* e vinculada ao cliente juntamente com o fornecimento do arquivo. O cliente a usará como prova do direito de usar o ativo. Caso a licença seja compartilhada ilegalmente, deve ser possível rastreá-la de volta ao cliente original, revelando o **traidor**. Isso corresponde ao mecanismo de Detecção de Traidor (TD).

Note que a **Confidencialidade dos Ativos** é importante tanto para os próprios provedores como para os clientes, que desejam que o ativo pelo qual estão pagando continue a ter valor no futuro. Assim, embora não seja possível prevenir completamente o compartilhamento ilegal de ativos, é importante que, enquanto os clientes agirem corretamente, o conteúdo dos ativos seja mantido seguro e não seja divulgado pelos protocolos desenvolvidos neste trabalho.

Além disso, é importante garantir que os protocolos e mecanismos desenvolvidos assegurem a descentralização e a concorrência justa, de forma que nenhuma entidade central e confiável seja necessária e nenhum provedor possa obter vantagem sobre sua concorrência por meio de manipulação da rede.

3.2. Modelo de Ameaça

As primeiras ameaças imediatas derivadas dos atores apresentados consistem no cliente ou no provedor tentando obter vantagens sem cumprir suas obrigações no protocolo, caso em que serão chamados de **cliente fantasma** ou **provedor fantasma**, respectivamente. Devem existir mecanismos para prevenir ou punir essas ações usando o ambiente público da *blockchain*.

Outra ameaça mencionada anteriormente é o **traidor**, um cliente que compartilha os ativos ilegalmente, contra o qual desejamos ter os mecanismos de TD. O cliente também pode tentar se passar pelo provedor na *blockchain* após obter um ativo, agindo como um **provedor não autorizado**. Além disso, um provedor poderia tentar manipular a *blockchain* para não validar transações de seus concorrentes comerciais, o que chamaremos de **provedor desonesto**, e é necessário um mecanismo de consenso capaz de prevenir essas ações.

Finalmente, consideraremos verificadores sob o modelo ***honest-but-curious* (HBC)** [Paverd et al. 2014], o que significa que executam as verificações de pagamento e

download corretamente, porém tentam obter o máximo de informações possível de cada execução do protocolo. Esse modelo é escolhido porque o design da *blockchain* incluirá mecanismos de consenso que garantem ***Byzantine Fault Tolerance*** **(BFT)**, ou seja, um atacante deverá obter o controle da maior parte do poder computacional da rede para manipular a verificação de maneira eficaz em um ambiente público [Xu et al. 2023]. Além disso, a existência de verificadores HBC significa que precisamos de um protocolo de *zero-knowledge* que não revele nenhuma nova informação sobre o conteúdo dos ativos a partir das provas de *download*.

Com os atores, requisitos e modelo de ameaça, a próxima seção se concentrará no desenvolvimento do protocolo criptográfico para provisionamento de ativos com provas de *download*. Posteriormente, esses protocolos serão incorporados no design da *blockchain* pública.

4. Verificando *downloads* - *Provable Data Provisioning* (PDPr)

Das seções anteriores, deriva-se que é necessário um protocolo para provisionamento de *download* que permita a emissão de uma prova de *download* verificável pelo provedor. Este protocolo deve ser de *zero-knowledge*, ou seja, não deve revelar nenhuma nova informação sobre o ativo em cada execução. Essa seção busca desenvolver um protocolo que atenda a essa propriedade de forma eficaz e performática.

Este será chamado de protocolo de *Provable Data Provisioning (PDPr)*, pois é similar ao *Provable Data Possession* (PDP) [Walker et al. 2022], exceto que o nó que recebe os dados não os possui *imediatamente*, ou seja, primeiro deve fornecer uma assinatura usada como parte da prova do provedor para o verificador. Assumiremos que um *identificador* do ativo é previamente conhecido publicamente, e será basicamente o resultado da aplicação de alguma função *hash* criptográfica ao ativo.

4.1. PDPr com *hash* homomórfico - uma primeira abordagem

O *hash* homomórfico (HomHash) é uma ferramenta criptográfica que permite o cálculo do *hash* de alguns dados após uma operação sobre os dados sem a necessidade de recalcular o *hash* do resultado da operação.

Uma proposta de HomHash é fornecida em [Bellare and Micciancio 1997] através do conceito de *incrementalidade*. Eles propõem um conjunto de construções que tornam possível obter um algoritmo de *hash* homomórfico sob a concatenação dos dados com novos dados. Considere um arquivo F e alguns dados gerados N inseridos aleatoriamente em F, resultando em um novo *arquivo adulterado* T. A construção deles permite obter um HomHash com a propriedade de que

$$HomHash(T) = HomHash(F) + HomHash(N) \tag{1}$$

É possível notar que, se $HomHash(T)$, $HomHash(F)$, e N são conhecidos, pode-se verificar a relação entre esses ao computar

$$HomHash(N) \stackrel{?}{=} HomHash(T) - HomHash(F) \tag{2}$$

Com essa ferramenta, um protocolo de PDPr pode ser construído:

1. O provedor gera uma semente DRBG (*Deterministic Random Bit Generator*) única.
2. O provedor usa a semente para inserir sequências aleatórias de bytes em lugares aleatórios no arquivo e fornece o arquivo adulterado ao cliente.
3. O cliente calcula o HomHash do arquivo recebido, assina o resultado e o envia de volta ao provedor.
4. O provedor envia a semente DRBG junto com a assinatura do cliente para o verificador.
5. O verificador checa a relação entre T, F e N com a relação (2).
6. O cliente recupera a semente DRBG do verificador e a usa para calcular o que deve ser removido do arquivo adulterado para recuperar o arquivo original.

Este protocolo é amplamente inspirado pelo ***Proof-of-Download*** **(PoDl)** [Costa et al. 2022]. É também um protocolo de *zero-knowledge proof*, pois possui as propriedades de *completude*, *corretude* (*soundness*) e *zero-knowledge*.

No entanto, tem a desvantagem de exigir que o verificador calcule todos os *nonces* que foram inseridos no arquivo, o que pode requerer mais memória e computação no caso de arquivos maiores, tornando seu uso em *smart contracts* inviável devido às altas taxas que isso incorre. Também exige que o provedor gere e distribua os arquivos adulterados para cada cliente interessado, aumentando a carga sobre o provedor.

Existe ainda um ataque que compromete a propriedade de **Confidencialidade dos Ativos** mencionada na seção 3.1. Considere um atacante A que tenta recuperar o arquivo original a partir de cópias adulteradas sem receber a semente DRBG do provedor:

1. A solicita ao provedor N versões adulteradas do arquivo. Ele pode usar múltiplos endereços *blockchain* e endereços IP para evitar ser detectado como um único cliente.
2. A lê as cópias e faz a interseção de seus blocos de dados, removendo os blocos que não são comuns.
3. A continua solicitando cópias adulteradas, removendo mais blocos em cada execução até que uma tentativa de força bruta para remover os blocos restantes e recuperar F se torne viável.

Com este ataque, A pode recuperar o arquivo original sem fornecer prova ao provedor, possivelmente não pagando pelo ativo. Isso quebra a propriedade de Confidencialidade dos Ativos. Assim, embora um protocolo semelhante se ajuste ao modelo de ameaça para PoDl, ele não se ajusta ao modelo de ameaça considerado aqui. Portanto, outras abordagens são necessárias, sendo apresentadas a seguir.

4.2. PDPr por meio de criptografia homomórfica

A Criptografia completamente homomórfica (*Fully Homomorphic Encryption* - FHE) [Acar et al. 2018] tem sido a base para vários novos avanços na comunidade de criptologia. Entre esses, a possibilidade de avaliar algoritmos de *hash* seguros sobre dados cifrados tem sido explorada [Bendoukha et al. 2022], e PDPr pode ser uma aplicação possível. Em resumo, a ideia é obter um par de algoritmo de cifração E e *hash* criptográfico H tal que, omitindo os detalhes da avaliação homomórfica, tenha a propriedade

$$H(E_K(F)) = E_K(H(F)) \tag{3}$$

Suponha que o provedor e o cliente possuam um esquema de FHE confiável e um algoritmo de *hash* criptográfico que possa ser avaliado sob este esquema FHE. O protocolo seria o seguinte:

1. O provedor gera uma chave única para o FHE e a utiliza para cifrar os dados. Em seguida, envia os dados cifrados para o cliente.
2. O cliente avalia homomorficamente o *hash* sobre os dados cifrados, assina o resultado e envia esta assinatura de volta para o provedor.
3. O provedor verifica a assinatura recebida e depois a publica para o verificador junto com a chave.
4. O verificador checa o *hash* recebido, decifrando-o com a chave.
5. O cliente recupera a chave através do verificador e a utiliza para decifrar os dados recebidos anteriormente.

Este protocolo aproveita a capacidade do FHE de permitir operações sobre dados cifrados sem exigir a decifração, garantindo assim que o *hash* possa ser avaliado sem revelar o conteúdo dos dados originais.

Assumindo que o esquema FHE e o algoritmo de cifração são seguros, o cliente não será capaz de recuperar os dados cifrados durante a avaliação do *hash*. Além disso, assumindo que o algoritmo FHE escolhido não revela o histórico de computações feitas sobre os dados, o verificador não será capaz de recuperar os dados originais a partir de seu *hash*, compondo a propriedade de *zero-knowledge* esperada.

A desvantagem deste método é que, para arquivos volumosos, o processo de avaliação se torna inviável. Para contornar essa limitação, propomos cifrar o arquivo com um cifra simétrica segura, como AES, e distribuir o arquivo cifrado publicamente, aplicando PDPr apenas sobre a chave de criptografia simétrica, que será muito menos volumosa (por exemplo, 256 bits para AES-256). Chamaremos esse processo de **minificação de ativos**.

No entanto, mesmo após a minificação, a avaliação ainda não é eficiente, conforme mostrado em pesquisas recentes utilizando esquemas FHE atuais, pois ainda é muito cara em termos de tempo [Bendoukha et al. 2022] para o caso de uso considerado aqui, que requer mais recursos computacionais para um ambiente de *blockchain* pública. Portanto, uma versão melhorada, eficiente e segura do protocolo anterior é apresentada na sequência.

4.3. PDPr com *hash* homomórfico - uma abordagem aprimorada

Considere **LtHash**, de [Bellare and Micciancio 1997], como uma implementação do HomHash. É baseado em reticulados (*lattices*) inteiras para garantir a segurança. Em particular, pode ser vista como uma função *one-way* $LtHash : \mathbb{Z} \rightarrow \mathbb{Z}_p^n$, onde p é o módulo usado em LtHash. Este algoritmo pode ser aproveitado como uma sub-rotina que permite a construção de um par de hash criptográfico e um esquema de cifração que interagem de forma semelhante ao protocolo anterior baseado em criptografia homomórfica.

4.3.1. Esquema de cifração - GCrypt

Sejam $\mathcal{M} = \mathcal{K} = \mathcal{C} = \mathbb{Z}_p^l$, respectivamente, espaços de mensagens, chaves e cifras. Seja também $GCrypt = (Enc, Dec, Gen)$ um esquema de cifração com $Enc : \mathcal{M} \times \mathcal{K} \rightarrow \mathcal{C}$

definido por

$$Enc(m, k) = m + k \tag{4}$$

onde a soma é feita sobre o grupo obtido pela soma direta de $\mathbb{Z}_p$ groups. A função de decifração $Dec : \mathcal{C} \times \mathcal{K} \to \mathcal{M}$ é obtida pela operação inversa. Se Gen escolher chaves uniformemente de $\mathcal{K}$ com probabilidade uniforme, então esse esquema possuirá *perfect secrecy*, de forma similar ao *One-Time Pad* (OTP).

4.3.2. Hash criptográfico - GHash

Considere LtHash como $LtHash : \mathbb{Z} \to \mathbb{Z}_p^n$ para a construção de um algoritmo $GHash : \mathbb{Z}_p^l \times \mathbb{Z}_p^n \to \mathbb{Z}_p^n$ definido por

$$GHash(m, m_\epsilon) = m_\epsilon + \sum_{i=1}^{l} m_i \times LtHash(i) \tag{5}$$

Onde usamos as entradas de m como multiplicadores para o LtHash de seus índices, tomando a operação $\times$ como a multiplicação escalar no espaço vetorial $\mathbb{Z}_p^n$. É possível notar que o somatório se reduz a uma multiplicação de matrizes.

O valor m_ϵ representa um termo de erro adicionado ao resultado da soma. Observa-se que a reversão de GHash sem o conhecimento de m_ϵ se reduz a resolver uma instância de *Learning With Errors* (LWE), que se presume ser computacionalmente difícil. Além disso, pode-se afirmar que GHash exibe uma propriedade homomórfica:

$$GHash(a + b, a_\epsilon + b_\epsilon) = GHash(a, a_\epsilon) + GHash(b, b_\epsilon) \tag{6}$$

Ao aplicar esse algoritmo ao esquema GCrypt apresentado anteriormente, é possível notar que, se GCrypt e GHash utilizarem os mesmos módulos, então

$$GHash(c, c_\epsilon) = GHash(m, m_\epsilon) + GHash(k, k_\epsilon) \tag{7}$$

Onde $c = Enc(m, k)$ e $c_\epsilon = m_\epsilon + k_\epsilon$. Claramente, GCrypt compartilha as mesmas desvantagens que o esquema de *One-Time Pad (OTP)*: as chaves nunca devem ser usadas mais de uma vez e a chave precisa ter o mesmo tamanho que a mensagem a ser cifrada. Para contornar essas limitações, é proposto usar a *minificação de ativos* mencionada anteriormente. A chave simétrica precisa ser codificada para o espaço de mensagens $\mathcal{M}$, o que pode ser feito simplesmente codificando bits para números ímpares ou pares escolhidos uniformemente de $\mathbb{Z}_p$. Apesar de ter um alto fator de expansão, quando aplicado a uma chave de 256 bits ainda será viável transferi-la entre o provedor e o cliente.

Com isso, podemos obter um protocolo PDPr da seguinte forma:

1. O provedor cifra simetricamente o ativo com uma chave e publica o ativo cifrado junto com o GHash da chave codificada, $GHash(m, m_\epsilon)$, para o qual escolhem um inteiro aleatório r_m e usam $m_\epsilon = LtHash(r_m)$ como erro, **mantendo-o em segredo**.
2. O provedor gera outra chave de cifração única k com uma semente DRBG k_{seed} (que deve ser curta para propagação em *blockchain*) e outro erro $k_\epsilon = LtHash(r_k)$. Calcula $c = Enc(m, k)$ e $c_\epsilon = k_\epsilon + m_\epsilon$. Em seguida, o provedor envia (c, c_ϵ) para o cliente.

3. O cliente calcula e depois assina o GHash do texto cifrado recebido, usando c_ϵ como erro.
4. O provedor publica a semente DRBG k_{seed} e k_ϵ junto com a assinatura como prova de fornecimento para o verificador, que utiliza a propriedade 7 para checar.
5. O cliente obtém a semente DRBG com o verificador e a utiliza para gerar k e recuperar $m = Dec(c, k)$.

O cliente não pode decifrar o texto cifrado sem a chave obtida do verificador, pois isso equivaleria a quebrar um esquema com segredo perfeito. Além disso, enquanto o verificador pode checar que uma determinada chave foi usada para cifrar a mensagem, ele não pode recuperar o conteúdo do texto cifrado a partir de seu GHash, nem recuperar o erro m_ϵ a partir das informações recebidas na prova, impedindo a recuperação da mensagem m.

5. Verificando pagamentos - interoperabilidade com outra *blockchain*

O modelo proposto também considera os pagamentos como um aspecto importante do DRM. Incorporar pagamentos em uma nova *blockchain* não é uma decisão de design ideal porque (1) requer dos usuários a confiança em uma nova moeda apenas para executar o provisionamento de ativos; e (2) aumenta o acoplamento entre pagamentos e *downloads*, criando riscos e exigindo um mecanismo de consenso mais complexo.

Por isso, propõe-se que uma *blockchain* bem estabelecida, como *Ethereum* [Buterin et al. 2014], seja utilizada para os pagamentos, enquanto os *downloads* acontecem em uma *blockchain* projetada como uma ***sidechain* com *Simple Payment Verification* (SPV)** [Wang 2021, Belchior et al. 2021]. Isso significa que a *blockchain* observará outras *blockchains* de criptomoedas populares existentes para verificar pagamentos ou *smart contracts*, utilizando algum tipo de *relay* confiável para tornar essa verificação mais eficiente por meio de cache de informações.

Apesar de requerer a implementação de interoperabilidade, isso ajuda na adoção geral da solução pela comunidade e permite maior flexibilidade. Chamaremos a *blockchain* usada para pagamentos de **cadeia de pagamentos** e a *blockchain* usada para *downloads* de **cadeia de downloads**.

6. Design da *blockchain*: mecanismo de consenso e forja de blocos

Agora, os aspectos de alto nível da solução *blockchain* são apresentados, abstraindo a incorporação dos protocolos de verificação de *download* e pagamento. Essa incorporação, que define quando e como os pagamentos, *downloads* e suas verificações ocorrem, será chamada de **esquema de download** e será abstraída por enquanto para permitir um design mais flexível.

6.1. A popularidade dos provedores

O Proof-of-Download (PoDl) [Costa et al. 2022] se baseia na ideia-chave de que cada provedor possui uma **popularidade** associada que é calculada a partir do número de *downloads*. O mesmo conceito é utilizado aqui para ajudar a proteger os clientes contra um *provedor não autorizado*. Um cliente que busca um provedor de um ativo de interesse pode verificar a popularidade do provedor antes de realizar pagamentos. Para

uma contagem eficiente de *downloads*, essas podem ser verificadas contra as provas de *download* e utilizadas pelos nós da *blockchain*. A popularidade também será utilizada nos esquemas de *download* para punir um *provedor fantasma*.

6.2. Mecanismo de consenso - baseado no consenso de Nakamoto com *Proof-of-Work* (PoW)

A presença de possíveis nós de provedores desonestos, ou seja, nós que tentam prejudicar comercialmente seus concorrentes manipulando a *blockchain*, significa que o mecanismo de consenso não pode ser baseado em *Proof-of-Stake* (PoS), pois isso provavelmente favoreceria os provedores maiores e reduziria o incentivo para que os provedores menores usem a *blockchain*. Portanto, um algoritmo de consenso inspirado em *Proof-of-Work* (PoW) do consenso proposto por Nakamoto [Nakamoto 2008] é proposto.

No entanto, para tentar reduzir as chances de um nó específico de provedor controlar grande parte do poder de mineração e usar isso como vantagem, o PoW original é modificado para formar um algoritmo de *proof-of-work* com limiar, chamado aqui de *threshold PoW*. Ou seja, em vez de requerer apenas uma solução de quebra-cabeça criptográfico, são necessárias pelo menos $Min_{verifiers}$ soluções de diferentes mineradores. As soluções formarão um comitê, onde cada solução conta para a inclusão de suas transações, e uma transação será incluída no próximo bloco se receber a maioria ($Min_{verifiers}/2+1$) dos votos. Com isso, espera-se que mesmo controlando grande parte do poder da *blockchain*, os provedores desonestos façam mais esforço para controlar votos suficientes neste comitê para manipular as transações.

A *blockchain* é implementada como uma *sidechain* de uma cadeia de pagamentos. Portanto, os mecanismos para recompensar os mineradores devem garantir que, após a verificação do pagamento, eles incluam uma recompensa aos mineradores, definida por cada esquema de *download*. Além disso, a recompensa deve ser proporcional ao esforço útil realizado por cada minerador para o bloco forjado, o que significa que os mineradores que votaram em transações que não foram incluídas devem receber menos recompensas, pois não verificaram efetivamente as transações.

Por fim, é importante definir como lidar com bifurcações *forks* no design do algoritmo de consenso [Xu et al. 2023]. Aqui, propõe-se que a cadeia escolhida seja aquela que possui a maior agregação de popularidade $P_{agg} = \sum_{t \in T} Popularity(t_{provider})$, onde T é o conjunto de transações da cadeia. Um valor maior de P_{agg} favorece a permanência de grandes provedores, que serão os mais populares e que em sua maioria "pagarão a conta"para os mineradores. Esse mecanismo também previne que provedores desonestos dominem a *blockchain*, já que eles teriam que acumular transações suficientes para criar uma cadeia com mais popularidade do que a agregação de transações em outras cadeias no *fork*. Isso é difícil porque, além de incluir as transações desses provedores desonestos, as outras cadeias no *fork* também incluirão transações de outros provedores potencialmente muito populares.

7. Propostas de esquemas de *download*

Com os aspectos de alto nível da *blockchain* apresentados, discutimos agora algumas propostas de **esquemas de download**, que definem como os *downloads* e pagamentos são incorporados (*embedded*) na *blockchain* além de quando e como eles ocorrem e são

verificados. Provas de pagamento e *download* são fornecidas como transações pelo cliente e pelo provedor, e os nós da *blockchain* interagem por meio dessas transações.

7.1. Esquema Pay-then-Download

Esta primeira proposta consiste no cliente primeiro pagar pelo ativo, apresentar a prova de pagamento e, somente então, fazer o *download* do ativo. A rede verificará a prova de pagamento e, após sua inclusão em um bloco, o provedor executará o *download* e emitirá uma prova de *download*. Se um provedor não fornecer uma prova de *download* dentro de n blocos após a verificação do pagamento, ele será considerado um *provedor fantasma*. Assim, sua popularidade será reduzida como punição para evitar que outros clientes realizem pagamentos ao mesmo provedor.

Para recompensar os mineradores, cada pagamento deve incluir a recompensa para os mineradores do n-ésimo bloco antes do atual, onde n está relacionado ao tempo de confirmação da transação. Ao verificar as transações, os mineradores devem aceitar apenas transações com prova de pagamento se estas incluírem essa recompensa. Cada minerador tem incentivo para fazer essa verificação porque, após n blocos, eles também devem ser recompensados.

Este método é simples e não requer *smart contracts*, o que permite taxas de execução mais baixas na cadeia de pagamento. Além disso, é muito eficiente na verificação de transações e uma boa opção para tempos menores de confirmação de transações. No entanto, exige confiança no provedor.

7.2. Esquema Lock-then-Prove

Para reduzir o risco de um pagamento ser feito pelo cliente sem a provisão do *download*, uma ideia é usar um *smart contract* para garantir que o pagamento só ocorrerá após a apresentação de uma prova de *download* válida. Para isso, introduzimos o conceito de *zero-knowledge time locked contracts*.

7.2.1. Zero-knowledge time locked contracts

A técnica **Hash Time Locked Contracts** (HTLC) [htl 2021] é utilizada para implementar interoperabilidade entre *blockchains*. No HTLC, os ativos a serem trocados entre as *blockchains* são bloqueados por meio de *smart contracts* que requerem a revelação de um segredo para desbloqueá-los dentro de um limite de tempo. Modificando a revelação do segredo, pode-se construir um protocolo semelhante para trocar recursos por informações confidenciais, permitindo ainda a verificação pública e a execução do *smart contract*.

Suponha que Bob queira obter informações confidenciais K de Alice em troca de criptomoeda. Alice revela a Bob alguma informação, P, da qual Bob não consegue diretamente derivar K. Bob usa essas informações para derivar uma prova P_{proof}, que não permite a recuperação de P e o codifica em um *smart contract* solicitando a revelação de alguma outra informação, R, a partir da qual ele possa construir a informação K e que pode ser verificada utilizando P_{proof}. Bob notifica Alice sobre este contrato, e Alice revela R publicamente. Com *zero-knowledge proofs*, a rede pode verificar que Alice revelou K para Bob usando P_{proof} e R, sem realmente obter conhecimento de K.

O que torna este problema difícil é que o *smart contract* deve ser baseado em informações que são verificáveis **publicamente**, mas as informações reveladas, diferentemente do HTLC, são **confidenciais**. Isso será chamado de **Zero-knowledge time-lock contract** (ZKTLC).

7.2.2. Lock-then-prove utilizando ZKTLC

A ideia é criar um ZKTLC que funcione como os verificadores de *download* previamente apresentados, codificando a verificação da prova de *download* no *smart contract* com o mesmo algoritmo utilizado pelos nós da *blockchain* durante a verificação.

O protocolo começará com o provedor fornecendo uma cópia cifrada ou alterada do ativo, dependendo do protocolo PDPr usado, e então o cliente criará um *smart contract* que requer a revelação das informações necessárias para recuperação do ativo original pelo provedor para desbloquear criptomoeda dentro de um limite de tempo, usando informações derivadas da cópia recebida do ativo, que será a assinatura fornecida pelo cliente ao provedor nos protocolos PDPr apresentados.

Neste novo protocolo, não é preciso confiar no provedor. Se o provedor estiver interessado em vender seu ativo, ele terá que participar adequadamente do ZKTLC. Não há necessidade de reverter o pagamento ou de mecanismos de punição, e o cliente tem a garantia de que sua moeda só será desbloqueada se a prova de *download* correta que permite a recuperação do ativo for revelada. Além disso, trabalhando apenas com *zero-knowledge proofs*, a Confidencialidade do Ativo é mantida. Os mecanismos de recompensa serão os mesmos propostos em 7.1.

No entanto, a grande desvantagem do esquema *lock-then-prove* é que a implementação dos protocolos PDPr com *zero-knowledge* apresentados é complexa e usa algoritmos que não estão nativamente implementados nas possíveis cadeias de pagamento. Isso faz com que as taxas de execução do contrato aumentem muito rapidamente.

7.3. Lock-then-prove com validação externa

Para utilizar o esquema de *lock-then-prove* com taxas de transação mais baixas, pode-se aproveitar o poder computacional dos mineradores da cadeia de *download* para verificar as *zero-knowledge proofs* e produzir *smart contracts* que verifiquem apenas as soluções dessas provas.

O cliente primeiro faz o *download* do ativo alterado ou cifrado e cria um *smart contract* que requer que $Min_{verifiers}$ endereços diferentes revelem suas soluções para o quebra-cabeça criptográfico da cadeia de *download*. Se pelo menos $Min_{verifiers}/2 + 1$ verificarem que a prova está correta, então a transação é desbloqueada. Os mineradores também recebem as recompensas automaticamente após essa confirmação.

Agora o *smart contract* é muito mais seguro, e os mineradores têm mais certeza de que receberão sua recompensa após validar um *download*. Além disso, isso aumenta a chance de que a transação correspondente ao ZKTLC seja publicada no próximo bloco.

8. Implementação: resultados e discussão

Os algoritmos foram implementados utilizando a linguagem Golang. Para aritmética modular de números inteiros grandes, optamos por implementar nosso próprio conjunto de ferramentas usando aritmética rápida de 64 bits em vez de usar o pacote de números inteiros grandes padrão da linguagem para melhorar amplamente o desempenho.

Todos os resultados apresentados foram obtidos com um processador Intel Core i7 de 7ª geração com 2.9GHz de clock e máquina com 8GB de RAM, comprovando a eficiência dos algoritmos propostos sem necessitar de hardware muito caro.

8.1. Implementação do LtHash

LtHash é definido em [Bellare and Micciancio 1997] e é usado como sub-rotina para GHash. Usamos o XOF (Função de Saída Expansível) do BLAKE2b [Aumasson et al. 2013], um algoritmo *hash* rápido e seguro, como função de randomização, dividindo a mensagem em blocos e aplicando o XOF para gerar um vetor de pedaços de tamanho fixo em bits. Conformes recomendado em [Bellare and Micciancio 1997], escolhemos 500 *chunks* de 128 bits para fornecer segurança e preservar o desempenho.

8.2. Implementação de GHash e GCrypt: resultados

Para o GHash, a implementação LtHash foi utilizada como sub-rotina, necessitando apenas do acréscimo da operação de inserção de bloco com multiplicação por um escalar utilizando grandes inteiros modulares. GHash usa os bytes da mensagem como *blocos* para serem multiplicados como escalares ao LtHash do índice de cada bloco. Os *benchmarks* apresentados na tabela 1 consideram blocos de 128 bits. Para o GCrypt, a primeira

Tabela 1. Resultados do *benchmark* do GHash

Tamanho de entrada (blocos)	Tempo (ms)
128	15.55
256	30.63

etapa da criptografia consiste na codificação de uma mensagem, que lê cada bit e gera um número ímpar ou par com base em seu valor usando um gerador de números aleatórios (*Pseudorandom Number Generator* - PRNG) não determinístico. A segunda etapa é a expansão da chave, que usa uma variante do HASH-DRBG [Barker and Kelsey 2015] baseada na iteração do SHA-256 com a chave como semente. A chave expandida é adicionada à mensagem codificada para obter o texto cifrado. Os *benchmarks* para cifração e decifração são apresentados na tabela 2

Tabela 2. Resultados de *benchmark* do GCrypt

Tamanho de entrada (bits)	Tempo de cifração (ms)	Tempo de decifração (ms)
128	1.130	0.580
256	2.280	1.152

8.3. Protocolo PDPr

O protocolo PDPr utilizando GHash e GCrypt oferece simplicidade sem perder eficiência e segurança. Além da cifração e decifração do ativo, o PDPr também utiliza a geração de provas, que corresponde ao cálculo do GHash sobre o ativo minificado cifrado no cliente, e a verificação de provas, que os verificadores de *download* executam. Os *benchmarks* para estas operações são apresentados na tabela 3.

Tabela 3. *Benchmarks* do protocolo de PDPr

Operação	Tamanho de chave	Tempo (ms)
Generação de prova	128/256	17.59/34.94
Verificação de prova	128/256	289.5/576.5

9. Conclusão e trabalhos futuros

Conforme apresentado, as contribuições são relevantes como as soluções distribuídas propostas para DRM e TD utilizando *blockchains* públicos e transparentes sem comprometer a eficiência e o desempenho. Pesquisa futura focará em aprimorar a definição e formalização das propriedades de segurança de protocolos de *Provable Data Provisioning* e aprimoramento de esquemas de *download*, por exemplo, formas de reduzir as taxas de execução na cadeia de pagamento dos contratos num esquema *lock-then-prove* sem depender de verificadores externos, uma vez que a implementação dos protocolos blockchain e o consenso ainda estão em aberto.

Os *benchmarks* apontam que a solução proposta seria viável em uma aplicação real, possivelmente escalável. Entretanto, a aplicação a ambientes reais envolve investigações adicionais, como pagamento de *royalties*, armazenamento de metadados, mecanismos de licenciamento mais fortes e adaptação a modelos de negócios específicos [Ciriello et al. 2023].

Referências

(2021). Hash Time Locked Contracts. `https://en.bitcoin.it/wiki/Hash_Time_Locked_Contracts`. (accessed September 14, 2023).

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.

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.

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 20q5, http://nvlpubs. nist. gov/nistpubs/SpecialPublications/NIST. SP*.

Belchior, R., Vasconcelos, A., Guerreiro, S., and Correia, M. (2021). A survey on blockchain interoperability: Past, present, and future trends.

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.

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.

Buterin, V. et al. (2014). A next-generation smart contract and decentralized application platform. *white paper*, 3(37):2–1.

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.

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.

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)*.

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

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*.

Paverd, A., Martin, A., and Brown, I. (2014). Modelling and automatically analysing privacy properties for honest-but-curious adversaries. *Tech. Rep*.

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

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.

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

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