GITI-CB: Gestão de Identidade com Troca de Informações entre Correntes de Blocos

Lucas Airam C. de Souza, Gabriel Antonio F. Rebello, Gustavo F. Camilo, Miguel Elias M. Campista, Luís Henrique M. K. Costa

[1]Grupo de Teleinformática e Automação (GTA)
Universidade Federal do Rio de Janeiro (UFRJ)

Resumo. *A identificação dos cidadãos é essencial para liberar o acesso a serviços básicos como saúde e educação. Entretanto, sistemas tradicionais de gestão de identidade adotam abordagens pouco escaláveis ou prejudiciais à segurança e à privacidade do usuário. Este artigo apresenta a proposta de um sistema de gestão de identidade baseado na interoperabilidade de correntes de blocos. No sistema, cada domínio mantém uma corrente de blocos para gerir a identidade de seus usuários. Além disso, o trabalho analisa a técnica de troca de informações entre correntes de blocos para implantar o conceito de "traga a sua própria identidade". Dessa forma, o sistema mitiga problemas de escalabilidade e garante aos usuários o controle sobre os seus dados.*

Abstract. *Citizen identification is essential to unlock access to basic services such as healthcare and education. However, traditional identity management systems adopt approaches that are poorly scalable or detrimental to user security and privacy. This paper presents a proposal for an identity management system based on blockchain interoperability. In the system, each domain maintains a blockchain to manage the identity of its users. Furthermore, the paper analyzes the technique of exchanging information between blockchains to implement the concept of "take your own identity". In this way, the system mitigates scalability problems and guarantees users control over their data.*

1. Introdução

A identidade é um dos bens mais preciosos do cidadão. Somente a partir de documentos que comprovam sua identidade, as pessoas conseguem obter acesso a direitos básicos como voto, educação, saúde e moradia, além de acesso a serviços e emprego. Apesar da extrema relevância para o bem-estar geral, aproximadamente 1 bilhão de pessoas no mundo não possuem ainda acesso a qualquer forma de prova de identidade [Pangestu et al. 2022]. Essa defasagem de registro acontece principalmente porque os processos de geração de documentos são altamente burocráticos e centralizados, criando empecilhos que aumentam os gastos e dificultam o acesso para parte da população. A centralização do gerenciamento de identidade afeta também os cidadãos que possuem registros, que confiam a guarda de seus dados a armazenamentos governamentais centralizados. Essa estratégia gera um ponto único de falha, implicando múltiplas vulnerabilidades de segurança para os dados dos usuários, que passam a estar sujeitos, por exemplo, a vazamento de informações e indisponibilidade devido a ataques de negação de

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001, CNPq, FAPERJ (E-26/203.211/2017, E-26/202.932/2017, E-26/202.689/2018 e E-26/200.892/2021), FAPESP (15/24494-8, 18/23292-0, 15/24485-9 e 14/50937-1.) e RNP (PGId)

serviço (*Denial of Service* - DoS). Mesmo esquemas de gerenciamento de identidades federados, que permitem o reaproveitamento de identidades, não são completamente descentralizados e, portanto, são vulneráveis a pontos únicos de falha. Dessa forma, garantir uma maneira segura e desburocratizada de criação e prova de identidade é essencial para reverter essa situação.

A tecnologia de corrente de blocos (*blockchain*) se apresenta como uma alternativa que pode ser utilizada para criar identificações descentralizadas (*Decentralized Identifiers* - DID) digitais simples e acessíveis em relação aos processos burocráticos comuns. Apesar do uso de corrente de blocos, as identidades podem ser limitadas em um provedor de serviços como a Amazon ou Google. Dessa forma, cada identidade criada possui validade apenas no domínio do seu provedor de serviços de criação, desconhecida por outros domínios. Assim, a cada novo serviço de que um usuário deseja usufruir, é necessário o cadastro e memorização das informações de acesso. Além disso, a dificuldade de memorização das informações para autenticação e acesso aos serviços motiva os usuários a tomarem medidas prejudiciais à segurança, como a reutilização de dados, armazenamento de informações em texto em claro e em locais inseguros. Dessa forma, a implementação de um sistema de identidade baseado em corrente de blocos que requer dos usuários o armazenamento de poucas informações para acessar diversos serviços é conveniente. Para atender a esse cenário, este artigo propõe um sistema de gestão de identidade em que as identidades são registradas em um domínio de forma segura utilizando uma corrente de blocos local e facilmente transportada para outro domínio através da comunicação por uma corrente de blocos global com múltiplos domínios.

Este artigo propõe uma arquitetura que aplica o conceito de "traga sua própria identidade" (*Bring Your Own Identity* - BYOI) através da troca de informações entre correntes de blocos. O sistema possui dois tipos de correntes de blocos: local e global. As correntes de blocos locais armazenam metadados de identidades dos clientes [Dunphy e Petitcolas 2018]. Assim, quando um cliente cria uma conta no domínio A, o domínio armazena os dados do cliente em seu próprio servidor e inclui o *hash* das informações cadastradas na corrente de blocos local. Dessa forma, a proposta garante ao usuário a integridade no registro de suas informações e que seus dados não são públicos na corrente de blocos local, reduzindo o custo de armazenamento de informações e mitigando problemas de privacidade. Caso o usuário queira se cadastrar em outro domínio, o domínio B emite uma transação à corrente de blocos global com uma requisição assinada pelo usuário. Os domínios monitoram a corrente de blocos global e verificam que uma transação foi emitida com o seu identificador como destino. Os domínios podem verificar a assinatura do usuário e autenticar o pedido.

O artigo está organizado da seguinte forma. A Seção 2 apresenta os trabalhos relacionados. A Seção 3 descreve a arquitetura proposta, enquanto a Seção 4 define o modelo de atacante adotado. Já a Seção 5 apresenta a implementação da arquitetura por um protótipo prático. A Seção 6 discute como o protótipo pode ser ampliado para um modelo de atacante mais robusto por meio da tecnologia SGX. Por fim, a Seção 7 conclui o artigo e apresenta direções futuras para o trabalho.

2. Trabalhos Relacionados

A mudança de paradigma trazida pela Web 3.0 torna necessária a existência de sistemas capazes de prover e gerenciar identidades de forma descentralizada. Recentemente, os DIDs tornaram-se um padrão de recomendação da *World Wide Web Consortium* (W3C) [Sporny et al. 2022]. Assim, há um número crescente de sistemas e plataformas propostos para atender a esses requisitos. O surgimento de propostas utilizando diferentes tecnologias, como o Sovrin [Reed et al. 2016] e uPort [Naik e Jenkins 2020], cria espaço para discussão e análise das soluções implementadas [Kondova e Erbguth 2020].

Xavier *et al.* discutem o uso de correntes de blocos na implementação de identidade autossoberana (*Self-Sovereign Identity* - SSI) e credenciais verificáveis [Xavier et al. 2021]. Os autores apresentam desvantagens da incorporação da corrente de blocos em uma arquitetura de registro de dados verificáveis (*Verifiable Data Registry* - VDR) em relação a bancos de dados centralizados, como o alto custo energético ao utilizar Prova de Trabalho (*Proof of Work* - PoW). Entretanto, existem outros protocolos de consenso, tornando a corrente de blocos adaptável às necessidades dos responsáveis pela manutenção do sistema. O protocolo de consenso *Practical Byzantine Fault Tolerance* (PBFT) é uma alternativa energeticamente eficiente, adotada por correntes de blocos permissionadas, enquanto a Prova de Posse (*Proof of Stake* - PoS) é uma alternativa de protocolo de consenso energeticamente eficiente para corrente de blocos não permissionadas.

O uPort [Naik e Jenkins 2020] é um sistema de gestão de identidade descentralizado desenvolvido sobre a plataforma Ethereum. O sistema é desenvolvido através de um contrato inteligente, sendo de fácil uso e implantação. Já o ChainID [Queiroz et al. 2021] é uma plataforma para gestão de identidades descentralizadas através da tecnologia de corrente de blocos. A proposta apresenta o uso da infraestrutura de identidades digitais em diversos fluxos de oferta de serviços, de forma segura e transparente. O Sovrin [Reed et al. 2016] é uma plataforma pioneira para criação de identidades digitais. A plataforma Sovrin contribuiu para o surgimento de diversas propostas após a doação de seu código-fonte como parte do projeto Hyperledger Indy. O Hyperledger Indy é um arcabouço de código-aberto que provê ferramentas necessárias para a criação de sistemas de identidade digitais [Hyperledger 2022]. Atualmente, diversos pesquisadores propõem soluções de identidades digitais que utilizam o Hyperledger Indy.

Bhattacharya *et al.* propõem um novo modelo para identidades autossoberanas utilizando o arcabouço Hyperledger Indy [Bhattacharya et al. 2020]. A proposta tenta mitigar ataques de homem no meio (*man-in-the-middle*) utilizando credenciais verificáveis. Além disso, os autores adicionam um fator de reputação para cada identidade gerada no sistema com base no usuário que a emitiu. O BrEduDevice é um mecanismo de gestão de identidades para a colaboração entre instituições na criação de sistemas de detecção de intrusão federado [Neto et al. 2021]. Os autores propõem um novo esquema de protocolo leve de acesso a diretório (*Lightweight Directory Access Protocol* - LDAP) para identificar e autenticar dispositivos participantes da federação.

As propostas anteriores visam um cenário com uma única corrente de blocos para a gestão de identidades. Essa abordagem possui dois problemas principais: a escalabilidade do sistema é afetada pela corrente de blocos única e os domínios perdem controle interno sobre o gerenciamento das identidades. A proposta atual prevê o uso de múltiplas correntes de blocos, classificadas como corrente de blocos global ou corrente de blocos local. As correntes de blocos locais são permissionadas e privadas, administradas pelos domínios provedores de serviços, enquanto a corrente de blocos global é mantida de forma pública para consultas às correntes de blocos locais e eventuais disputas entre as partes envolvidas. Dessa forma, o sistema é escalável, pois os domínios realizam a maioria das operações em paralelo e de forma independente, e economiza espaço de armazenamento, porque os serviços possuem acesso a informações essenciais para seu funcionamento. Além disso, o trabalho atual complementa o sistema apresentado em um trabalho anterior [de Souza et al. 2022].

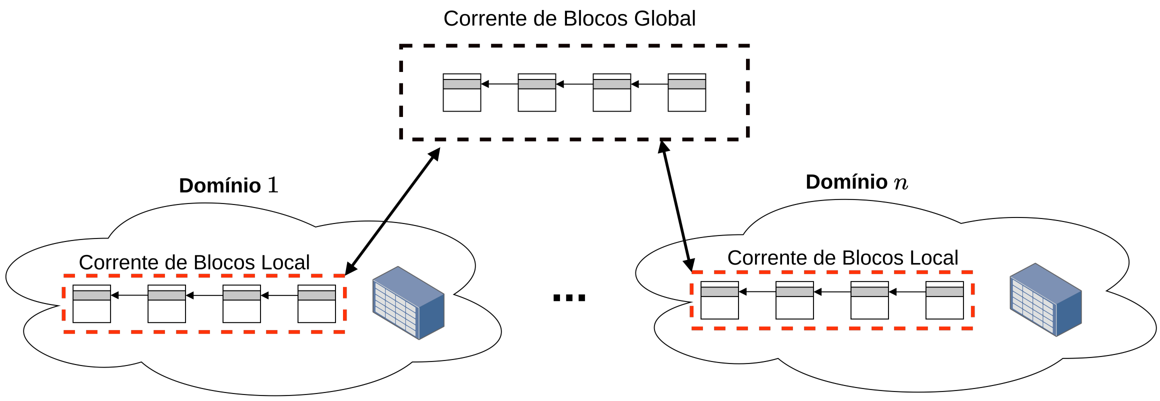

Figura 1: Arquitetura proposta para o sistema de gerenciamento de identidades.

3. Arquitetura Proposta

O cenário da proposta compreende múltiplos domínios administrativos concorrentes, como Facebook e Google, que gerenciam identidade dos usuários que utilizam seus sistemas. A Figura 1 ilustra a arquitetura proposta para o sistema. As correntes de blocos fornecem algumas propriedades, como imutabilidade, transparência e descentralização, necessárias para a garantia de segurança em um cenário com diversos participantes sem confiança mútua [Queiroz et al. 2021, de Souza et al. 2020]. Essa arquitetura permite uma alta escalabilidade, uma vez que cada domínio armazena dados somente de seu interesse, e privacidade, já que um domínio possui informações somente de seus clientes, sem acesso às informações de outras correntes de blocos locais.

A corrente de blocos global é responsável por registrar as requisições de informações de identidades entre domínios. A corrente de blocos global realiza as operações necessárias para garantir a interoperabilidade entre as correntes de blocos locais e transferência de informações requisitadas por um domínio a algum outro domínio.

As correntes de blocos locais são responsáveis por registrar informações dos usuários nos domínios provedores de serviços. Essas correntes são administradas pelos domínios, que podem escolher mecanismos de consenso e parâmetros de blocos mais rápidos e eficientes, além de registrar permissões e registro de seus clientes, sem revelar informações a outros domínios. Cada domínio possui total autonomia para gerenciar as próprias correntes de blocos, podendo privilegiar diferentes aspectos de acordo com suas preferências e demandas. As informações de registro de identidade são armazenadas para garantir transparência aos clientes. Para garantir a privacidade dos clientes, dados pessoais são armazenados fora da corrente. O registro, no entanto, apresenta o *hash* desses dados de maneira pública, provendo integridade da informação armazenada.

O objetivo da arquitetura proposta com uma corrente de blocos por domínio é mitigar problemas de escalabilidade com o número de usuários, evitando a existência de uma única corrente de blocos responsável pelo armazenamento de todos os dados. A escalabilidade é alcançada através da paralelização do gerenciamento de identidades por domínios diferentes em correntes de blocos locais distintas. Entretanto, há um compromisso para o usuário quanto ao modelo de segurança incorporado pela corrente de blocos local de seu provedor de serviço. Enquanto sistemas com uma única corrente de blocos mantêm maior resiliência quanto aos participantes maliciosos, correntes de blocos locais permissionadas com poucos participantes no mecanismo de consenso são mais vulneráveis. Além disso, as correntes de blocos públicas devem ser armazenadas por todos os participantes, apresentando uma sobrecarga. A proposta atual mitiga essa sobrecarga dividindo a informação em múltiplas correntes de blocos, assim os participantes armazenam apenas as informações contidas em correntes de blocos de seu interesse.

3.1. Correntes de Blocos Locais

As correntes de blocos locais são permissionadas e gerenciadas por cada domínio, que adota parâmetros e políticas específicas para seus clientes. Essa autonomia permite que os domínios adotem protocolos de consenso mais rápidos, escaláveis e mais adequados às suas necessidades. As correntes global e locais, no entanto, possuem funções bem definidas dentro da arquitetura, registrando identidades dos clientes de forma pública e sem ferir a privacidade de dados. Para o registro das identidades, este trabalho adota o padrão definido pela *World Wide Web Consortium* (W3C), organização internacional de padrões Web, de identificadores digitais descentralizados (*Decentralized Identifiers* - DID) [Sporny et al. 2022].

A transação de registro $T_{reg_{u,A}}$ é utilizada por um cliente u para registrar uma nova identidade em um domínio A. Essa transação é emitida a cada vez que o usuário u transfere um conjunto de dados pessoais $D_u = \langle d_1, d_2, ..., d_n \rangle$ para o domínio A. Este trabalho assume que os domínios adotam políticas de verificação de identidades para o registro de um usuário, como validação de documentos ou prova de identificação. Ao publicar essa transação na corrente de blocos, o usuário u possui total transparência das operações envolvendo seus dados armazenados por A. Para preservar a privacidade dos dados, a transação de registro $T_{reg_{u,A}}$ contém apenas o *hash* $H(D_u)$ dos dados e não os dados em si. Essa alternativa de publicar somente o *hash* dos dados também ajuda a reduzir problemas relacionados a escalabilidade e armazenamento das correntes de blocos, uma vez que somente o metadado das informações são publicadas. Além disso, a publicação do *hash* em uma estrutura imutável como a corrente de blocos garante a integridade dos dados. Assim, caso o usuário u descubra que algum de seus dados foi alterado no futuro, ele pode conferir que eles foram corretamente publicados na corrente de blocos a partir de $H(D_u)$ e tomar uma ação contra o domínio A. Por fim, u também define um conjunto de permissões $P_u = \langle p_1, p_2, ..., p_m \rangle$ que o domínio A pode executar com seus dados, como compartilhar ou comercializar com outros domínios. Assim, é possível definir a transação de registro $T_{reg_{u,A}}$ como:

$$T_{reg_{u,A}} = [\text{Sig}_u | \text{Sig}_A | H(D_u) | P_u | \text{DID}_u], \tag{1}$$

em que Sig_u é a assinatura do usuário u autorizando o registro da identidade, Sig_A é a assinatura do domínio A concordando que o usuário u está registrado em seu domínio, $H(D_u)$ é o *hash* dos dados registrados por u, P_u é o conjunto de permissões que o domínio A possui sobre os dados de u e DID_u é o DID gerado. Todas as transações são assinadas tanto pelo domínio quanto pelo usuário para garantir o não-repúdio da operação.

3.2. Corrente de Blocos Global

Assim como as correntes de blocos locais, a corrente de blocos global também é permissionada. Entretanto, as duas correntes diferem-se quanto ao conteúdo que é armazenado e o gerenciamento da corrente de blocos. Enquanto as correntes de blocos locais são gerenciadas por cada domínio, que estabelece regras próprias quanto ao consenso e entrada de participantes, a corrente de blocos global é gerenciada pelos domínios em conjunto. Assim, cada domínio participante do sistema age como validador da corrente de blocos global. Os clientes também participam da corrente de blocos local monitorando as transferências de seus próprios dados de identificação entre os domínios, porém sem participar ativamente da validação de transações. Além disso, o principal objetivo da corrente global é armazenar as transferências de dados e identidade dos clientes entre diversos domínios, como no modelo de "traga sua própria identidade", mantendo a transparência para o usuário cliente.

No modelo proposto, a transação de requisição, denotada por $T_{req_{u,A}}$, é utilizada por um domínio qualquer participante da rede de corrente de blocos global para requisitar dados do usuário u ao domínio A. Essa transação é emitida toda vez que um domínio, autorizado pelo usuário u, quer obter informações já registradas em correntes locais de outros domínios. Além disso, a transação contém um conjunto de chaves identificando os dados que o domínio pede para o outro domínio, que detém os dados. A transação é publicada na corrente de blocos, permitindo que o usuário verifique que o pedido foi efetuado corretamente e que o domínio de origem que armazena os dados possa verificar. A transação de requisição $T_{req_{u,A}}$ emitida por um domínio B para obter dados do usuário u armazenado pelo domínio A é definida por:

$$T_{req_{u,A}} = [\text{Sig}_u | \text{Sig}_B | DK_u | DID_u | \text{Dst}_A], \tag{2}$$

em que Sig_u é a assinatura do usuário u autorizando o pedido dos dados, Sig_B é assinatura do domínio B que faz a requisição dos dados, DK_u são as chaves dos dados, e.g. e-mail, telefone, que o domínio B quer acesso, DID_u é o identificador do usuário u e Dst_A é o domínio destino do pedido, i.e., o destino que possui os dados.

O domínio que possui os dados do usuário monitora a corrente de blocos global verificando se as transações são destinadas a ele. O domínio detentor, caso seja o destino do pedido, deve responder ao pedido emitindo uma transação de resposta positiva ou negativa ao domínio requerente. Apesar da separação entre resposta positiva ou negativa, as duas transações, para simplificar, compartilham a mesma estrutura. A diferenciação entre as duas é feita pela utilização dos campos das transações. A transação de resposta negativa é publicada quando o domínio detentor rejeita o pedido do domínio requerente e apresenta o erro contendo os motivos da rejeição. O principal motivo para que uma requisição possa ser rejeitada é por não corresponder ao conjunto de permissões cedidas pelo usuário na transação de registro. Dessa maneira, como a transação de registro contém as permissões e está armazenada na corrente de blocos global, um contrato inteligente pode facilmente verificar se a requisição é válida, sem a necessidade de intermediários. A transação de resposta deve também referenciar a transação de requisição correspondente para facilitar a identificação pelos participantes e a validação pelo contrato inteligente. Ademais, a transação de resposta, caso positiva, deve informar o ponto de acesso que B pode usar para se comunicar com A e obter os dados. A transação de resposta $T_{rep_{u,B}}$ emitida pelo domínio A ao domínio requerente B é definida por:

$$T_{rep_{u,B}} = [\text{ReqTxID} | \text{Rs}_{T_{req_{u,A}}} | \text{Er}_{T_{req_{u,A}}} | EP_A], \tag{3}$$

em que ReqTxID é o identificador da transação de requisição ao qual a transação de resposta se refere, $\text{Rs}_{T_{req_{u,A}}}$ é o resultado do pedido da transação de requisição, identificando se foi aceito ou negado, $\text{Er}_{T_{req_{u,A}}}$ expressa o erro do pedido, caso seja negado, e EP_A é o ponto de acesso que A disponibiliza a B. O campo que indica o ponto de acesso aparece em branco quando o pedido é negado.

A Figura 2 mostra o diagrama de sequência da arquitetura proposta, incluindo a transação de registro, os atores e as correntes de blocos de cada domínio, a corrente de blocos global e as interações entre todas as partes envolvidas. A seguir, cada passo ilustrado é detalhado brevemente:

1. o usuário compartilha seus dados e permissões com o administrador do domínio de origem, que irá registrar os dados em seus sistemas;

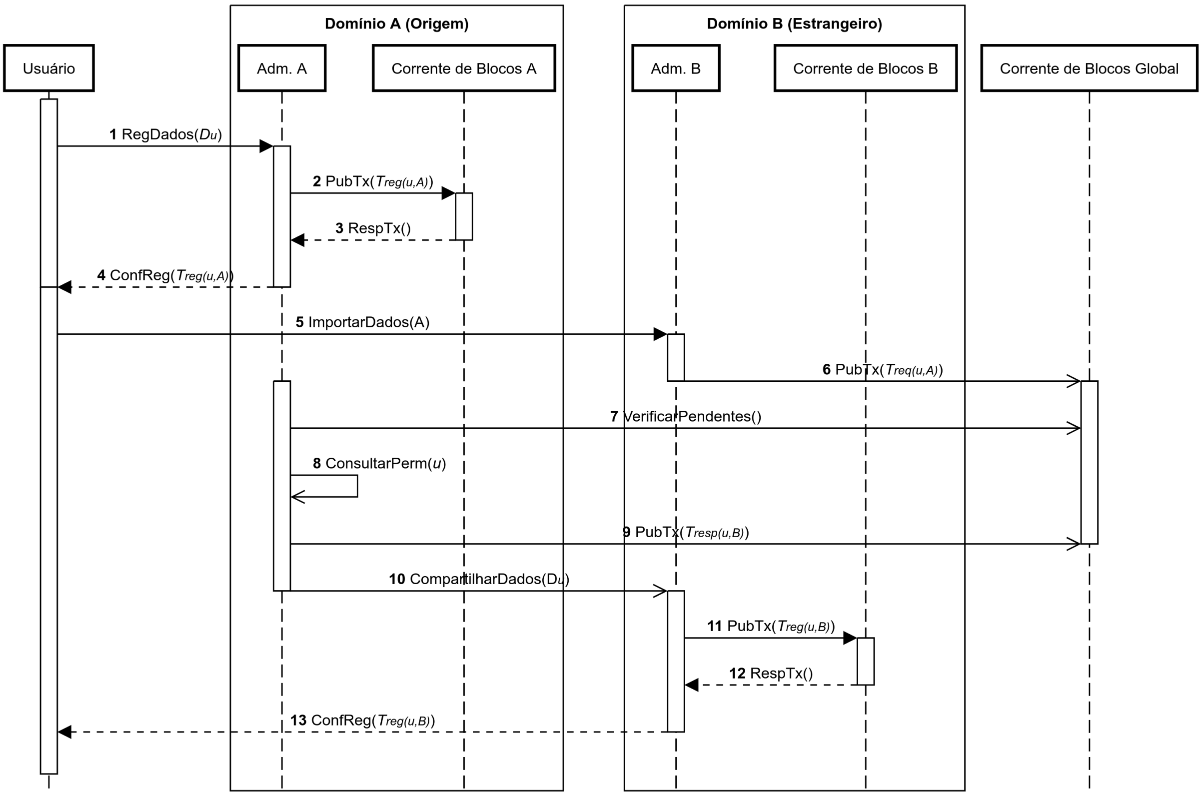

Figura 2: Diagrama de sequência da arquitetura proposta. Os administradores de cada domínio colaboram, através da corrente de blocos global, para importar os dados do usuário sem que seja necessário o reenvio de dados.

2. após o registro dos dados e permissões, o administrador publica a transação de registro $T_{reg_{u,A}}$ na corrente de blocos local;
3. uma vez publicado o bloco que inclui $T_{reg_{u,A}}$, a corrente de blocos responde ao administrador A;
4. o administrador A repassa a confirmação do registro ao usuário, que pode, a partir deste momento, verificar a corrente de blocos local para garantir que seus dados estão seguros;
5. ao se registrar em um domínio estrangeiro B, o usuário solicita a importação de dados do domínio A, sem necessidade de preencher novamente dados que já foram passados;
6. o administrador do domínio B registra o pedido de importação de dados através de uma transação de pedido $T_{req_{u,A}}$ e publica na corrente de blocos global;
7. em sua busca periódica na corrente de blocos global por novas solicitações ou a partir de uma requisição, o administrador A verifica que há um pedido do domínio B por dados do usuário contidos no domínio A;
8. o administrador A consulta em seu sistema as permissões relativas aos dados e usuário solicitados;
9. confirmada a permissão, o administrador A emite uma transação de resposta $T_{resp_{u,B}}$ registrando a autorização para o compartilhamento dos dados solicitados;
10. o administrador A comunica-se diretamente com o administrador B e compartilha os dados do usuário;
11. após o recebimento dos dados, o administrador B publica uma transação local de registro $T_{reg_{u,B}}$ em sua corrente de blocos local;
12. a corrente de blocos local confirma a publicação do novo registro;
13. finalmente, o administrador B responde ao usuário confirmando a importação de seus

dados. A partir deste momento, o usuário também pode verificar que os dados foram compartilhados corretamente verificando a corrente de blocos do domínio B.

As etapas 3 e 12 correspondem a simplificações da sequência de ações devido às restrições de espaço na figura. No diagrama completo, é necessário que cada administrador verifique ativamente suas correntes de blocos para confirmar a publicação da transação em questão.

4. Modelo de Atacante

Esta seção descreve o modelo de atacante considerado no sistema. O modelo de atacante é dividido em duas partes: o modelo de atacante inicial e o modelo de atacante com ambientes de execução confiáveis. O modelo de atacante inicial contempla integralmente a prova de conceito realizada no artigo.

4.1. Modelo de Atacante Inicial

Este trabalho utiliza o modelo "honesto, mas curioso" para especificar os ataques à nuvem que armazena os dados de um proprietário [Božović et al. 2012, Xu et al. 2013, Krämer et al. 2019, Xue et al. 2017]. Nesse modelo, a entidade comprometida por um atacante continua a seguir honestamente o protocolo do sistema. O interesse do atacante consiste em vazar informações sensíveis da organização ou dos clientes, e não comprometer a integridade dos dados ou o comportamento da entidade. Neste trabalho, as principais ameaças consideradas são intrusões ou ataques internos de funcionários, que têm como principal objetivo permanecerem não-descobertas, portanto, raramente alteram o comportamento da entidade. Por outro lado, para modelar o comportamento das organizações e dos clientes em suas interações com a corrente de blocos, este trabalho considera o modelo de atacante Dolev-Yao [Dolev e Yao 1983]. Este modelo prevê os atacantes mais poderosos em um cenário inseguro [Cervesato 2001], em que o atacante pode ler, enviar e descartar uma transação endereçada à corrente de blocos, ou qualquer pacote da rede. Ao contrário dos ataques à nuvem que armazenam as identidades, o atacante nesse cenário pode se conectar passivamente à rede e capturar trocas de mensagens ou injetar, reproduzir, filtrar e trocar informações ativamente.

Os ataques às entidades da arquitetura proposta são baseados em modelos existentes na literatura e descritos abaixo [Camilo et al. 2020, Alvarenga et al. 2018]:

Ataques à corrente de blocos objetivam impedir que uma transação ou bloco legítimo seja adicionado à corrente de blocos. O protocolo de consenso tolerante a falhas do sistema mitiga esse tipo de ataque, pois exige que o atacante controle uma parcela significativa da rede para afetar o protocolo de consenso. Caso contrário, o emissor da transação/bloco pode verificar a qualquer momento sua presença na corrente de blocos. Os ataques que exigem corrupção ou adulteração de transações são impossíveis quando todas as transações incluem seu *hash* correspondente assinado.

Ataques aos clientes e às organizações consistem em tentar obter informações privadas ou personificar o alvo. A arquitetura proposta permite a auditoria de todas as transações passadas. Portanto, se um invasor tentar modificar a corrente de blocos usando pares de chaves roubados, a tentativa é registrada. Após a descoberta de um incidente, o agente que sofreu o ataque pode facilmente substituir os pares de chaves roubados, restabelecendo a segurança e evitando mais danos. Este trabalho não aborda o caso em que um cliente age de forma deliberadamente maliciosa ao, por exemplo, mentir sobre os próprios dados. O trabalho assume que os domínios participantes utilizam mecanismos de verificação externa de dados.

Ataques à rede representam a tentativa de isolar um único alvo, impedindo assim que vendedores e compradores emitam transações ou que algum participante leia conteúdo da corrente de blocos. Esta categoria de ataque contempla ataques clássicos de rede, que podem ser mitigados através do estabelecimento de caminhos redundantes entre a corrente de blocos e organizações, clientes e proprietários. Este trabalho assume uma rede pública redundante, como a Internet, que interconecta todos os participantes.

5. Implementação de um Protótipo do Sistema Desenvolvido

Como prova de conceito, a arquitetura proposta foi implementada para verificar a viabilidade da solução[1]. A implementação é baseada na plataforma de código aberto Hyperledger Fabric 2.4 [Androulaki et al. 2018]. O aspecto organizacional do Fabric se ajusta ao cenário multi-domínio da arquitetura proposta e a possibilidade de divisão da corrente em múltiplos *canais* logicamente isolados permite a simulação de múltiplas corrente de blocos locais com diferentes características. O cenário executado na prova de conceito apresenta dois domínios que possuem sua própria corrente de blocos e efetuam o protocolo de troca de dados descrito na Seção 3. Além da implementação da corrente de blocos no Hyperldeger Fabric, também foram desenvolvidos códigos em Python que implementam a comunicação entre os domínios, clientes e corrente de blocos para uma prova de conceito.

5.1. Caso de Uso da Arquitetura Proposta

Primeiramente, seguindo os passos da Figura 2, o usuário registra os dados em um domínio que tenha interesse. Para isso, o usuário utiliza o comando `saveData` e passa suas informações através de um `socket` conectado ao servidor do domínio. O Código 1 ilustra o comando utilizado pelo usuário e o formato de mensagem que é enviado ao servidor na prova de conceito implementada. O usuário executa o programa em Python direcionado a ele, passando o próprio nome, e-mail e cadastro de pessoa física (CPF) para o domínio, além de utilizar o argumento `share` para passar informações que o domínio pode compartilhar com outros. No caso do exemplo do código, o usuário permite que o domínio compartilhe seu nome, CPF e e-mail com terceiros. Para garantir o isolamento entre as entidades simuladas, o código foi separado entre o cliente e os dois domínios, possuindo assim processos independentes. Os dados são enviados pela porta TCP 5041, em que o código do domínio está escutando, em notação de objeto JavaScript (*JavaScript Object Notation* - JSON) para facilitar o armazenamento, a busca de dados e uniformizar a troca de dados.

```
python3 client_main.py saveData --name Gabriel --cpf 000.000.000-00 --email
    gabriel@gta.ufrj.br --share 'name,cpf,email'
Saving data: {"name": ["Gabriel", "share"], "cpf": ["000.000.000-00", "
   share"], "email": ["gabriel@gta.ufrj.br", "share"]} through port 5041
Connecting to server
Sending data
```

Código 1: Execução do código do cliente para armazenar os dados em um determinado domínio. O cliente envia seus próprios dados, neste exemplo, nome, CPF e e-mail, além das permissões de compartilhamento pela interface de linha de comando (*Command Line Interface* - CLI).

Ao receber os dados do usuário, o servidor verifica se os formatos dos dados foram passados corretamente e emite uma transação de registro em sua corrente de blocos local, como descrito na Seção 3.1. O Código 2 ilustra o *log* do servidor ao receber os dados do usuário, o comando de emissão de uma transação de registro e o resultado da emissão.

[1]Disponível em https://github.com/GTA-UFRJ/giti-cb

```
python3 server_main_org1.py
Issuing command: peer chaincode invoke -o localhost:7050 --
    ordererTLSHostnameOverride orderer.example.com --tls --cafile $CAFILE --
    peerAddresses localhost:9051 --tlsRootCertFiles $TLS_FILES -c '{"
    function":"StoreData","Args":["{\"name\": \"share\", \"cpf\": \"share\",
     \"email\": \"sahre\"","uid1","usig1","upubkey1","Org1","HASH
    ","0.0.0.0:2000"]}'
```

Código 2: Emissão da transação de registro pelo domínio. No final do comando, a subdivisão "`Args`" mostra o que é armazenado na corrente de blocos local, como definido pela transação $T_{reg_{u,A}}$.

Após registrar seus dados em um domínio, o cliente pode facilmente verificar na corrente de blocos que uma transação de registro foi publicada com sua identidade. O cliente pode, então, transferir esse registro para outros domínios e facilitar novos registros. Para isso, o cliente utiliza o comando `issueRequest` comunicando ao domínio que deseja registrar seus dados utilizando o registro de outro domínio. Essas informações são enviadas para o novo domínio através de um `socket` implementado em Python para a porta TCP 4052. O usuário passa então informações como quais dados o domínio pode pedir ao outro e qual domínio armazena os dados. Ao receber o pedido do cliente, o domínio requerente, Org2, emite uma transação de requisição definida na Seção 3.2 na corrente de blocos global. O Código 3 exibe o resultado do comando executado pelo usuário.

```
python3 client_main.py issueRequest --data_keys name.cpf,email --dst_org
    Org1 --user_id uid1
Sending ['name', 'cpf', 'email']|Org1|uid1 through port 5042
Connecting to server
Sending data
```

Código 3: Execução do código do cliente que deseja registrar seus dados em um determinado domínio utilizando dados já armazenados em outro domínio. O cliente pede o compartilhamento do nome, CPF e e-mail armazenados no domínio Org1.

Por fim, o domínio que registrou os dados do usuário, Org1, verifica a transação e emite uma transação de resposta referenciando a transação de requisição. Essa transação de resposta pode ser facilmente verificada por qualquer participante da corrente de blocos global através de uma simples `query`.

6. Direções Futuras com Ambientes de Execução Confiáveis

Esta seção apresenta os ambientes de execução confiáveis e como a introdução dessa tecnologia pode estender o modelo de atacante atual. Primeiramente, a tecnologia é descrita de forma geral e apresenta-se o seu funcionamento por meio de um ambiente de execução confiável. Por fim, considera-se um modelo de atacante ampliado, que considera a introdução de ambientes de execução confiáveis no sistema proposto.

6.1. Ambiente de Execução Confiável com Enclaves

O Grupo de Computação Confiável (*Trusted Computing Group* - TCG) propôs o desenvolvimento de plataformas que fornecem segurança usando recursos de *hardware* para lidar com a crescente ameaça de ataques virtuais e vazamento de dados. Dentre essas plataformas destaca-se a tecnologia de Extensões de Proteção de Software (*Software Guard Extensions* - SGX) que protege dados e *softwares* mesmo se componentes com alto nível de privilégio forem comprometidos ou contaminados por códigos maliciosos.

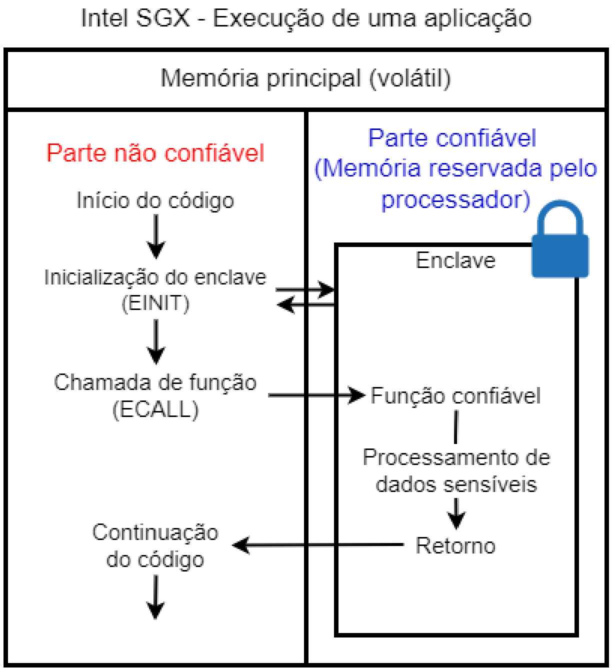

Figura 3: Exemplo de execução de uma aplicação utilizando a tecnologia SGX. A aplicação inicializa através da parte não confiável um enclave de memória (instrução `EINIT`) na parte confiável. Posteriormente, o acesso à parte confiável é realizado com chamadas de uma função confiável (instrução `ECALL`) que processa dados confidenciais em um ambiente isolado. O acesso é protegido pelo processador. Figura de Thomaz *et al.* [Thomaz et al. 2022].

Essa tecnologia é adequada para um modelo de atacante mais robusto, discutido na Seção 6.2, pois garante a execução confiável da Etapa 10 no diagrama exibido na Figura 2. Essa etapa é crítica, pois nela ocorre o compartilhamento dos dados entre domínios, portanto é necessário garantir que o domínio compartilha apenas os dados que o usuário permitiu e apenas com os domínios permitidos. Assim, o ambiente de execução confiável garante a confidencialidade e integridade quando a BIOS, o sistema operacional, o hipervisor ou o administrador da máquina são maliciosos. Dessa forma, o modelo de atacante pode contemplar provedores maliciosos de serviços que tenham intenção de vazar dados sensíveis fora da arquitetura proposta.

O SGX executa instruções especiais dos processadores Intel para criar Ambientes de Execução Confiáveis (*Trusted Execution Environments* - TEEs) e armazenar dados em enclaves. Os enclaves são regiões isoladas e encriptadas de memória, que só podem ser acessadas pelo proprietário do enclave utilizando instruções confiáveis [Costan e Devadas 2016]. Essa divisão é feita para utilizar funcionalidades como chamadas de sistema (`syscall`) e acesso a entradas e saídas, que só estão disponíveis fora de enclaves. A Figura 3 exibe como ocorre a execução de uma aplicação no Intel SGX, onde apenas os dados críticos de uma aplicação são executados dentro da parte confiável, reduzindo a quantidade de informações armazenadas no enclave [Will et al. 2017]. No sistema proposto, os dados dos usuários são as informações críticas que devem ser processadas em enclaves para garantir que sejam processadas corretamente. Além disso, a infraestrutura do SGX oferece o processo de selagem para armazenar os dados sensíveis de forma persistente. Neste processo, o conteúdo dos enclaves é criptografado por chaves exclusivas da CPU e o resultado é transferido para o disco, de forma que só possa ser decriptado na mesma plataforma [Anati et al. 2013].

A execução de um procedimento de atestação remota é indispensável para a arquite-

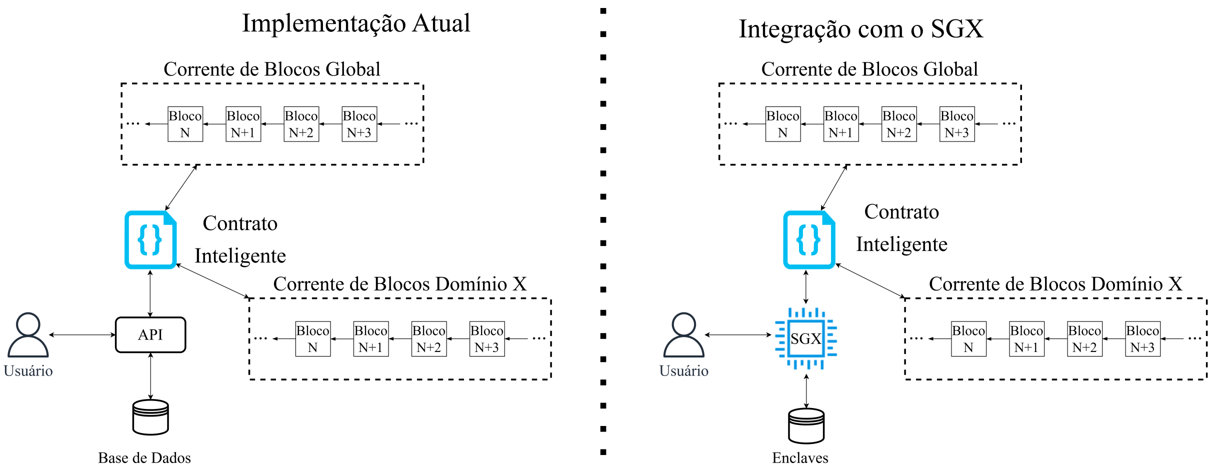

Figura 4: Proposta de integração do sistema com o SGX. À esquerda, a versão atual da prova de conceito. À direita, o novo esquema de execução após integração ao SGX.

tura proposta, pois o servidor que processa os dados em enclave precisa provar sua segurança e autenticidade para o cliente. Para isso, a Intel provisiona a cada processador uma chave de grupo chamada identificação de privacidade aprimorada (*Enhanced Privacy ID* - EPID), que só pode ser verificada ou revogada por uma entidade confiável chamada serviço de atestação da Intel (*Intel Attestation Service* - IAS) [Thomaz et al. 2022]. Na atestação, a CPU verifica o *hash* do código de um enclave e envia um relatório assinado para a parte interessada em verificar a segurança, que o encaminha para o servidor de atestação verificar a assinatura [Johnson et al. 2016]. A Intel oferece a possibilidade dos serviços de processamento de dados implementarem seu próprio sistema de verificação de atestação, utilizando certificados provisionados pela Intel ao invés da chave EPID [Scarlata et al. 2018]. O protocolo de atestação se baseia em troca de chaves *Diffie-Hellman* de curva elíptica entre cliente e enclave, para transmitir o relatório em um canal encriptado. Se a atestação for bem sucedida, o cliente envia os segredos por este canal seguro, de modo que apenas o enclave consiga decriptá-los.

A tecnologia SGX permite a verificação da Etapa 10, assegurando que os domínios compartilham apenas as informações solicitadas pelos clientes e com os respectivos domínios que solicitaram os dados de identificação do usuário. Ademais, os enclaves garantem que apenas o código do próprio enclave pode ter acesso aos dados. Logo, o único meio para compartilhá-los é através do sistema de correntes de blocos, segundo a lógica definida previamente. A Figura 4 exibe como o sistema é implementado atualmente e a sua respectiva versão integrada ao SGX, apresentada como uma direção futura a ser estendida em relação à prova de conceito atual.

6.2. Modelo de Atacante com Ambientes de Execução Confiáveis

O modelo de atacante nesta etapa considera como atacante o provedor de serviços ou um usuário malicioso que invadiu o sistema como descrito por Gueron [Gueron 2016]. O adversário considerado tem controle total sobre o sistema operacional e as aplicações em execução nele com um privilégio de qualquer nível, e pode ler ou modificar o conteúdo da DRAM, incluindo copiar e reproduzir dados. Assim, o interesse dos agentes consiste em compartilhar dados sensíveis de forma não autorizada e não seguir o protocolo previsto, para que o sistema não registre as ações maliciosas no sistema de correntes de blocos. Entretanto, assume-se que os agentes maliciosos são incapazes de fraudar algoritmos criptográficos. Os únicos componentes considerados confiáveis pelo SGX são os componentes internos da CPU.

7. Conclusão

Este artigo propôs o GITI-CB um sistema de gestão de identidades baseado no paradigma "traga a sua própria identidade". O sistema usufrui dos benefícios da tecnologia de corrente de blocos para criação de identidades de forma descentralizada através de uma solução hierárquica escalável com múltiplas correntes de blocos. A integração da tecnologia SGX à arquitetura atual do GITI-CB é proposta como trabalhos futuros, para incluir um modelo de atacante mais robusto. Além disso, a prova de conceito deve ser ampliada para um protótipo com outras correntes de blocos.

Referências

Alvarenga, I. D., Rebello, G. A. F. e Duarte, O. C. M. B. (2018). Securing configuration management and migration of virtual network functions using blockchain. Em *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, páginas 1–9.

Anati, I., Gueron, S., Johnson, S. e Scarlata, V. (2013). Innovative Technology for CPU Based Attestation and Sealing. *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, 13(7).

Androulaki, E. et al. (2018). Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. Em *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, New York, NY, USA. Association for Computing Machinery.

Bhattacharya, M. P., Zavarsky, P. e Butakov, S. (2020). Enhancing the Security and Privacy of Self-Sovereign Identities on Hyperledger Indy Blockchain. Em *2020 International Symposium on Networks, Computers and Communications (ISNCC)*, páginas 1–7. IEEE.

Božović, V., Socek, D., Steinwandt, R. e Villányi, V. I. (2012). Multi-Authority Attribute-based Encryption with Honest-but-Curious Central Authority. *International Journal of Computer Mathematics*, 89(3):268–283.

Camilo, G. F., Rebello, G. A. F., de Souza, L. A. C. e Duarte, O. C. M. B. (2020). AutAvailChain: Automatic and Secure Data Availability through Blockchain. Em *IEEE Global Communications Conference (GLOBECOM)*, páginas 1–6.

Cervesato, I. (2001). The Dolev-Yao Intruder is the Most Powerful Attacker. Em *16th Annual Symposium on Logic in Computer Science—LICS*, volume 1.

Costan, V. e Devadas, S. (2016). Intel SGX Explained. *Cryptology ePrint Archive*.

de Souza, L. A. C., Antonio F. Rebello, G., Camilo, G. F., Guimarães, L. C. B. e Duarte, O. C. M. B. (2020). DFedForest: Decentralized Federated Forest. Em *IEEE International Conference on Blockchain (Blockchain)*, páginas 90–97.

de Souza, L. A. C., Camilo, G. F., Rebello, G. A. F., Campista, M. E. M. e Costa, L. H. M. (2022). Gestao Segura e Escalável de Identidades através de Multiplas Corrente de Blocos. Em *Anais Estendidos do XXII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, páginas 167–170. SBC.

Dolev, D. e Yao, A. (1983). On the Security of Public Key Protocols. *IEEE Transactions on information theory*, 29(2):198–208.

Dunphy, P. e Petitcolas, F. A. (2018). A First Look at Identity Management Schemes on the Blockchain. *IEEE Security & Privacy*, 16(4):20–29.

Gueron, S. (2016). A Memory Encryption Engine Suitable for General Purpose Processors. *Cryptology ePrint Archive*.

Hyperledger (2022). Hyperledger Indy. Relatório técnico, Hyperledger Foundation.

Johnson, S., Scarlata, V., Rozas, C., Brickell, E. e Mckeen, F. (2016). Intel Software Guard Extensions: EPID Provisioning and Attestation Services. *White Paper*, 1(1-10):119.

Kondova, G. e Erbguth, J. (2020). Self-Sovereign Identity on Public Blockchains and the GDPR. Em *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, páginas 342–345.

Krämer, M., Frese, S. e Kuijper, A. (2019). Implementing Secure Applications in Smart City Clouds using Microservices. *Future Generation Computer Systems*, 99:308–320.

Naik, N. e Jenkins, P. (2020). uPort Open-Source Identity Management System: An Assessment of Self-Sovereign Identity and User-Centric Data Platform Built on Blockchain. Em *2020 IEEE International Symposium on Systems Engineering (ISSE)*, páginas 1–7. IEEE.

Neto, H. N. C. et al. (2021). BrEduDevice: Um Mecanismo Eficaz para a Gestão de Identidades para Sistemas de Detecção de Intrusão Federados. *XI Workshop de Gestão de Identidades Digitais (WGID)*.

Pangestu, M. et al. (2022). ID4D Data: Global Identification Challenge by the Numbers. Relatório técnico, The World Bank.

Queiroz, S., Greve, F., Sampaio, L. N. e Marques, E. (2021). Plataforma para Gestão de Identidades Descentralizadas Baseada em Blockchain. Em *Anais do XXI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, páginas 29–42. SBC.

Reed, D., Law, J. e Hardman, D. (2016). The Technical Foundations of Sovrin. *The Technical Foundations of Sovrin*.

Scarlata, V., Johnson, S., Beaney, J. e Zmijewski, P. (2018). Supporting third party attestation for intel sgx with intel data center attestation primitives. *White paper*.

Sporny, M. et al. (2022). Decentralized Identifiers (DIDs) v1.0. Relatório técnico, W3C Recommendation.

Thomaz, G. A. et al. (2022). CACIC: Controle de Acesso Confiável Usando Enclaves a Dados em Nuvem da Internet das Coisas. Em *Anais do XL Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, páginas 573–586. SBC.

Will, N. C., Condé, R. C. e Maziero, C. A. (2017). Mecanismos de segurança baseados em hardware: uma introdução à arquitetura intel sgx. *Minicursos do XVII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais. Brasilia, DF, BR: Sociedade Brasileira de Computação*, páginas 49–98.

Xavier, M. R., Cabral, K., Sette, I. S. e Ferraz, C. A. (2021). Sobre o uso de Blockchain em soluç oes com Credenciais Verificáveis e Identidades Auto-Soberanas. *XI Workshop de Gestão de Identidades Digitais (WGID)*.

Xu, J., Chang, E.-C. e Zhou, J. (2013). Weak Leakage-Resilient Client-Side Deduplication of Encrypted Data in Cloud Storage. Em *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, páginas 195–206.

Xue, K., Xue, Y., Hong, J., Li, W., Yue, H., Wei, D. S. e Hong, P. (2017). RAAC: Robust and Auditable Access Control with Multiple Attribute Authorities for Public Cloud Storage. *IEEE Transactions on Information Forensics and Security*, 12(4):953–967.