Submission date: 12/11/2025
Resubmission date: 17/02/2026
Camera ready submission date: 30/03/2026

1st round notification: 02/02/2026
2nd round notification: 25/03/2026
Available online: 03/04/2026
Publication date: 03/04/2026

Section: [Extended Version of XXV Simpósio Brasileiro de Cibersegurança (SBSeg 2025)]

Implementação e Análise de um Protocolo de Sincronização para Troca Justa com Justiça Forte e Privacidade.

Implementation and Analysis of a Synchronisation Protocol for Fair Exchange with Strong Fairness and Privacy

Dhileane Quixabeira[1], Mailson Teles-Borges[1], Fabricia Roos-Frantz[1], Rafael Z. Frantz[1], Sandro Sawicki[1]

[1]Unijuí University – UNIJUI
Ijuí, RS – Brazil

{dhileane.rodrigues, mailson.borges}@sou.unijui.edu.br, {frfrantz, rzfrantz, sawicki,}@unijui.com.br

Abstract. *In distributed information systems, it is essential to ensure fairness and privacy in the exchange of digital items between parties that do not fully trust each other. In such environments, multiple entities (users, organizations, services, devices, etc.) interact by exchanging data, documents, or digital assets. In the absence of trust between the parties involved, the risk of unfairness arises, namely the possibility that one party gains unfair advantage over the other. Fair exchange protocols have been developed to ensure that no participant obtains this advantage, thereby guaranteeing that exchanges occur fairly. Privacy, another fundamental property, prevents the leakage of confidential information. However, achieving both strong fairness and privacy simultaneously remains a challenge. Protocols widely used in information systems for online payments, e-commerce, and digital signatures ensure fairness but compromise privacy because they rely on mediating centralised third parties that directly access participants' data, including sensitive information. In this paper, we propose the implementation and analysis of a protocol that ensures strong fairness in distributed information systems, which ensures that none of the participants is left with both items and grants each party the ability to cancel the exchange. The model uses a split trusted third party, composed of two trusted execution environments (one for each participant) and a public bulletin board used solely for synchronisation. This architecture obviates the need of centralised intermediaries and preserves essential properties such as strong fairness, immediate and unilateral cancelation, and privacy. This work is an extended version of the paper presented at SBSeg 2025; it includes new experimental results, formal verification with the SPIN model-checker, and implementation improvements; therefore it contributes to the advancement of interoperable solutions that protect data privacy in distributed information systems.*

Cite as: Quixabeira, D., Teles-Borges, M., Roos-Frantz, F., Frantz, R. Z., & Sawicki, S. (2026). Implementation and Analysis of a Synchronisation Protocol for Fair Exchange with Strong Fairness and Privacy (Implementação e Análise de um Protocolo de Sincronização para Troca Justa com Justiça Forte e Privacidade). iSys - Journal of Information Systems, 19, 1:1–1:32. DOI: 10.5753/isys.2026.6859

Keywords. *Fair Exchange, Fair Exchange Protocols, Distributed Information Systems, Protocol Validation, Model-checking, Trusted Hardware.*

Resumo. *Em sistemas de informação distribuídos, é essencial garantir justiça e privacidade na troca de itens digitais entre partes que não confiam plenamente umas nas outras. Nesses ambientes, várias entidades (usuários, organizações, serviços, dispositivos, etc.) interagem entre si, trocando dados, documentos ou valores digitais. Quando não há confiança entre as partes envolvidas, surge o risco de injustiça, ou seja, a possibilidade de uma delas obter vantagem indevida sobre a outra. Protocolos de troca justa foram desenvolvidos para assegurar que nenhuma das partes obtenha vantagem indevida, garantindo que a troca ocorra de maneira justa. Privacidade, outra propriedade fundamental, impede o vazamento de informações confidenciais. No entanto, alcançar simultaneamente justiça forte e privacidade ainda é um desafio. Protocolos amplamente usados em sistemas de informação para pagamentos online, e-commerce e assinaturas digitais garantem justiça, mas comprometem a privacidade ao depender de terceiros centralizados que acessam diretamente os dados das partes, expondo informações sensíveis. Neste artigo, propomos a implementação e análise de um protocolo com justiça forte voltado a sistemas de informação distribuídos, o qual garante que nenhum dos participantes obtenha ambos os itens e concede a cada parte o poder de cancelamento. O modelo adota um terceiro confiável dividido, formado por dois ambientes de execução confiáveis (um para cada participante) e um public bulletin board usado apenas para sincronização. Essa arquitetura elimina intermediários centralizados e preserva propriedades essenciais, como justiça forte, cancelamento imediato e unilateral e privacidade. Este trabalho é uma versão estendida do artigo apresentado no SBSeg 2025, incluindo novos resultados experimentais, verificação formal com o SPIN model checking, e aprimoramentos de implementação; contribuindo para o avanço de soluções que protege a privacidade dos dados e interoperáveis em sistemas de informação distribuídos.*

Palavras-Chave. *Fair Exchange, Fair Exchange Protocols, Sistemas de Informação Distribuídos, Validação de Protocolo, Model-checking, Hardwares Confiáveis.*

1. Introdução

Protocolos de Troca Justa (*Fair Exchange Protocols* – FEPs) permitem que dois participantes troquem itens entre si de maneira justa, assegurando que nenhuma das partes obtenha vantagens indevidas. Esses itens podem ser digitais, como arquivos, chaves criptográficas ou confidenciais; físicos, como uma garrafa de vinho ou uma pizza [Molina-Jimenez et al. 2024]; ou ainda uma combinação de ambos. Em ambientes digitais, tais protocolos tornam-se particularmente relevantes para garantir que transações ocorram sem risco de fraude, interrupção oportunista ou disputas posteriores.

Diversos FEPs já foram propostos na literatura, oferecendo diferentes propriedades – entre elas, a justiça (*fairness*), considerada a mais fundamental [Asokan et al. 1997, Brickell et al. 1988, Asokan et al. 2000, Pinkas 2003,

Avoine and Vaudenay 2004, Zhang et al. 2024]. De modo geral, a justiça assegura que, ao término do protocolo – seja por execução normal ou por cancelamento – nenhuma das partes envolvidas fique em desvantagem. Protocolos que garantem justiça forte (*strong fairness*) são denominados *Strong Fair Exchange Protocols (SFEPs)*. Entretanto, muitos desses protocolos dependem de terceiras partes confiáveis centralizadas (*Trusted Third Parties – TTPs*), que atuam como mediadoras da troca e processam informações sensíveis [Pagnia and Darmstadt 1999]. Essa dependência pode comprometer a privacidade e introduzir pontos únicos de falha, representando um desafio significativo em sistemas distribuídos, especialmente quando envolvem dados sigilosos ou de alto valor [Markowitch et al. 2003, Fischer et al. 1985, Asokan et al. 2000].

Com o avanço de Ambientes de Execução Confiáveis (*Trusted Execution Environments – TEEs*), surgiram propostas que substituem TTPs monolíticos por arquiteturas distribuídas (*split TTPs*) [Molina-Jimenez et al. 2024]. Entre elas, destaca-se o protocolo *Fair Exchange Without Disputes (FEWD)* de Molina-Jimenez et al. (2024) [Molina-Jimenez et al. 2024], que especifica propriedades do protocolo FEWD como *strong fairness*, cancelamento unilateral imediato (*strong timeliness*) e preservação da privacidade. Entretanto, a proposta é apresentada predominantemente em nível conceitual, sem implementação funcional e sem validação formal sistemática que comprove rigorosamente o atendimento dessas propriedades.

Diante desse contexto, as seguintes questões de pesquisa emergem como problemas abertos que demandam investigação no âmbito de Sistemas de Informação:

QP1: Como alcançar um FEP que garanta *strong fairness*, cancelamento unilateral e privacidade em SIs distribuídos sem depender de um terceiro totalmente confiável?

QP2: Como a implementação do protocolo de sincronização pode contribuir para enfrentar desafios sociotécnicos de confiança, responsabilização e governança em SIs?

QP3: Quais são as implicações dessa abordagem para SIs que operam em ambientes organizacionais e ecossistemas digitais que demandam troca segura de ativos digitais?

QP4: Como validar formalmente que as propriedades reivindicadas são efetivamente satisfeitas pelo protocolo proposto?

Este artigo responde a essas questões ao implementar o protocolo FEWD, modelá-lo por meio de Máquinas de Estados Finitos (*Finite State Machines – FSMs*), formalizá-lo em *Promela* e verificá-lo por *model checking* com a ferramenta *SPIN*. Assim, demonstramos experimentalmente sua correção, implementabilidade e efetividade.

A principal contribuição deste estudo consiste na implementação funcional, validação formal e demonstração prática de um protocolo de sincronização que assegura *strong fairness*, cancelamento unilateral imediato e preservação da privacidade, conectando teoria e implementação verificável.

O restante deste artigo está estruturado da seguinte forma: a Seção 2 apresenta a terminologia e as definições. Na Seção 2.5, discutimos o protocolo de Troca Justa Sem Disputas, o FEWD, a especificação do protocolo de troca justa que inspirou este trabalho. A Seção 4 descreve a metodologia adotada, enquanto a Seção 5 detalha o pro-

cesso de verificação formal. Na Seção 6, descrevemos nossa implementação em Python das operações do FEWD. Na Seção 7, demonstramos como nossa implementação da operação de sincronização preserva a *strong fairness*, a privacidade e outras propriedades. A Seção 8 resume os trabalhos relacionados. Por fim, na Seção 9, compartilhamos nossa experiência de implementação e as ideias estimuladas pelos resultados obtidos.

2. Fundamentação Teórica

A troca justa, como tema de pesquisa, ganhou ampla atenção no início dos anos 1980, graças ao trabalho de Asokan [Asokan et al. 1997]. Desde então, diversos protocolos foram propostos e podem ser utilizados para a troca de itens de diferentes naturezas (por exemplo, digitais ou físicos) e sob diferentes propriedades. Os protocolos de troca justa são executados entre duas partes, digamos, Alice e Bob, que possuem, respectivamente, os itens `D_A` e `D_B`, com o objetivo de trocá-los. A propriedade mais fundamental dos FEPs é a justiça (*fairness*). Sob essa perspectiva, eles podem ser divididos em duas classes:

- **Strong FEPs:** garantem *strong fairness*, ou seja, ao término do protocolo, Alice fica na posse de `D_B` e Bob na posse de `D_A`, ou os itens permanecem com seus proprietários originais. Esses protocolos produzem apenas resultados justos e, portanto, impedem a ocorrência de disputas.
- **Weak FEPs:** garantem apenas *weak fairness*, ou seja, podem produzir resultados injustos, nos quais uma das partes fica com os dois itens e a outra não recebe nada. Assim, esses protocolos consideram a possibilidade de disputas, que são resolvidas por mecanismos de resolução executados separadamente, fora da execução normal do protocolo e que, muito provavelmente, envolvem humanos atuando como árbitros.

Em 1999, Pagnia e Gartner demonstraram que a troca justa é, no mínimo, tão difícil quanto o consenso [Pagnia and Darmstadt 1999]; portanto, a *strong fairness* não pode ser alcançada sem a participação de um TTP. O TTP atua como um controlador central, responsável por manter o invariante de que as FSMs de ambos os participantes alcancem os mesmos estados finais. A característica marcante dos *weak FEPs* é que eles podem ser implementados sem a necessidade de um TTP (ver Seção 8).

2.1. Propriedades Desejadas de Fair Exchange

Além da propriedade de *strong fairness*, a privacidade, a *timeliness* e a independência de *timeout* físico também são propriedades importantes para avaliar um protocolo [Asokan et al. 1997, Markowitch et al. 2003, Huang et al. 2014]. A privacidade garante que apenas Alice e Bob tenham acesso às informações sensíveis. A *timeliness* assegura que um participante, por exemplo Alice, possa cancelar o protocolo de forma unilateral e imediata, sem a necessidade de aguardar mensagens adicionais de Bob, seja diretamente ou indiretamente, por meio de um TTP. Vale esclarecer que, para cancelar, Alice pode precisar enviar ou receber uma mensagem gerada localmente pelo TTP. A independência de *timeout* físico ocorre quando o protocolo não utiliza relógios físicos para determinar o tempo limite. O desafio consiste em garantir essas propriedades sem comprometer a *strong fairness*. Por exemplo, a *timeliness* é trivial de implementar sem *strong fairness*. Alice poderia simplesmente abandonar o protocolo a qualquer momento e perder seu item.

2.2. As Quatro Operações Básicas em Fair Exchange

Qualquer protocolo de troca justa pode ser dividido em quatro operações básicas: *deposit*, *verify*, *synchronisation* e *release/restore* [Molina-Jimenez et al. 2024]. Descreveremos as operações do ponto de vista de Alice; as execuções de Bob refletem as de Alice.

Antes da execução dessas operações, um *handshake* é realizado por Alice e Bob para concordarem com os termos e condições da troca, incluindo a descrição dos itens. Depois disso, a operação *deposit* é realizada por Alice para entregar seu item a Bob. A operação *verify* é executada por Alice em relação ao item de Bob, a fim de confirmar que se trata do item esperado. A operação *synchronisation* é então executada para determinar se a troca concluída ou cancelada. Por fim, executa-se a operação *release/restore* ou nenhuma ação. A operação *release* é realizada para entregar o item de Bob a Alice quando o protocolo é concluído com sucesso. Entretanto, a execução da operação *release* pode ser seguida da execução da operação *restore*, a fim de devolver o item de Alice (ou uma compensação) a ela, quando o protocolo é concluído por cancelamento. Em alguns protocolos, como o FEWD, o cancelamento não gera nenhuma operação; ambos os itens permanecem bloqueados indefinidamente nos TTPs.

Todos os protocolos de troca justa incluem essas operações, embora nem sempre estejam claramente explicitadas ou separadas. A abordagem utilizada para implementar e executar essas operações determina quais propriedades (ver Seção 2.1) o protocolo será capaz de oferecer. Por exemplo, nos que utilizam liberação gradual, a sincronização é realizada entre os dois participantes, sem o uso de um TTP; portanto, protocolos de liberação gradual só podem garantir *weak fairness*. De forma semelhante, naqueles baseados em *escrow*, a operação *deposit* é executada em um TTP *stateful*; consequentemente, esses protocolos não conseguem garantir privacidade.

2.3. Arquitetura: TTP Monolítico versus Dividido

Uma característica marcante dos *strong FEPs* existentes é que eles implementam seus TTPs seguindo uma arquitetura monolítica, como ilustra a Figura 1a; normalmente, o TTP é um servidor com recursos de armazenamento e computação. Nesta arquitetura, o TTP é utilizado para executar as quatro operações básicas discutidas na Seção 2.2. O destaque em negrito indica o local onde cada operação é realizada em cada uma das duas arquiteturas. O ponto forte desse tipo de arquitetura é a sua simplicidade e a familiaridade; contudo, ela compromete a privacidade.

Outra alternativa é a arquitetura dividida apresentada na Figura 1b. O TTP é composto por três componentes: dois *attestables* (um para cada participante) e um (*Public Bulletin Board – PBB*). Essa arquitetura distribui a confiança entre as partes, uma vez que cada componente é controlado por seu respectivo proprietário, reduzindo a dependência de uma entidade central. Os *attestables* são TEEs, como descrito na Seção 2.4. Eles são representados com linhas duplas para enfatizar sua camada de segurança. Esses componentes podem estar incorporados ao hardware do dispositivo de cada participante ou hospedados em outro local. O PBB pode ser implementado em qualquer servidor público (por exemplo, um servidor web ou uma plataforma de rede social). Sua funcionalidade é semelhante à da arquitetura blackboard [Colletti 2017]. Na nossa implementação, o

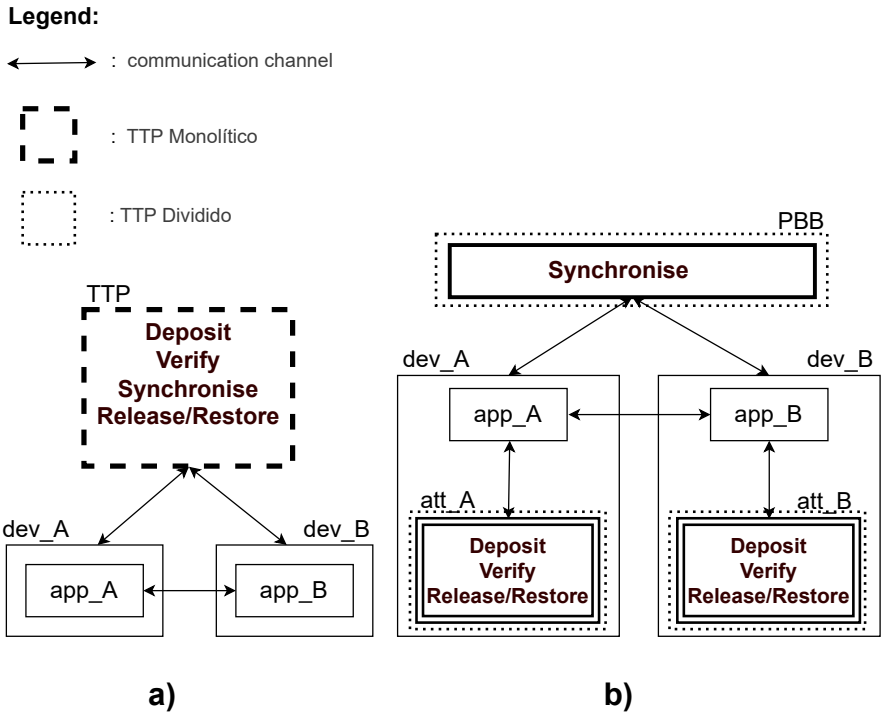

Figura 1. (a) TTP Monolítico Tradicional (b) TTP Dividido

principal requisito para o PBB é que ele seja publicamente acessível, ofereça uma API que aceite operações do tipo `post token`, possua um log de tamanho arbitrário para armazenar tokens permanentemente em uma determinada ordem e aceite operações do tipo `retrieve tokens` para disponibilizar todo o log. Um token é uma sequência de caracteres, por exemplo, “Oi”.

A vantagem dessa abordagem está na modularidade. Como mostrado na Figura 1b, a execução das quatro operações básicas (destacadas em negrito) pode ser cuidadosamente distribuída entre os três componentes para atingir diferentes propriedades. Essa é a abordagem adotada pelo FEWD, que será detalhada na Seção 2.5.

2.4. O Modelo Attestable e Tecnologias de Implementação Atuais

Um *attestable* é um modelo projetado para executar dados sensíveis de forma segura [Molina-Jimenez et al. 2024]. Ele oferece uma interface de programação (API) e deve atender a três propriedades fundamentais: (i) operar como um ambiente de execução isolado, no qual dados e computações permanecem ocultos do ambiente externo; (ii) garantir que, uma vez iniciado, o programa siga fielmente sua lógica, preservando a integridade e a independência da execução; e (iii) ser capaz de comprovar que essas propriedades foram efetivamente mantidas, por meio de mecanismos de atestação.

Um *attestable* pode ser implementado utilizando diferentes tecnologias de hardware e software. Entre elas, destacam-se *ARM TrustZone* [Pinto and Santos 2019],

Intel SGX [Costan and Devadas 2016], *AMD SEV* [Kaplan et al. 2016], *Amazon Nitro Enclaves* [Lutsch et al. 2025] e os compartimentos criados em *Morello Boards* [Grisenthwaite et al. 2023]. Essas soluções vêm se tornando cada vez mais comuns em dispositivos móveis, servidores e plataformas de nuvem.

2.5. Arquitetura do FEWD e as Quatro Operações Básicas

Esta seção aborda a QP1, apresentando a arquitetura de um protocolo de sincronização chamado FEWD que garante *strong fairness* e cancelamento unilateral e privacidade.

O FEWD é um *strong FEP* que utiliza um TTP dividido (ver Figura 2), apoiando-se em conceitos discutidos em [Molina-Jimenez et al. 2024], que orientam a implementação desse tipo de protocolo. Aqui, apresentamos apenas um resumo, focando nos conceitos necessários para compreender a implementação discutida na Seção 6.

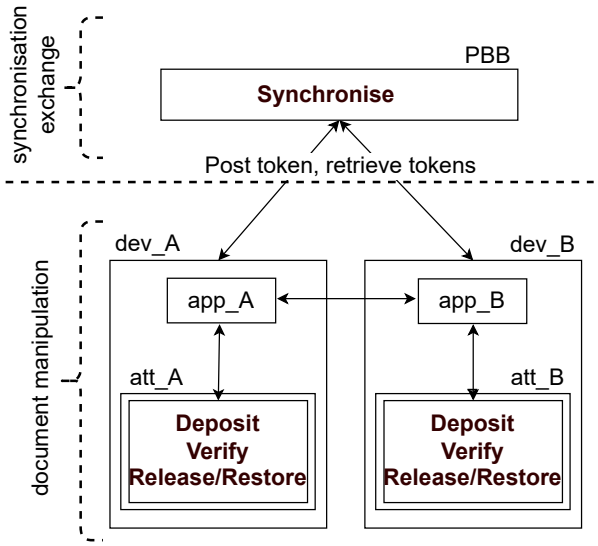

Figura 2. Arquitetura do FEWD composta por dois protocolos

Alice e Bob estão em posse de dispositivos pessoais `dev_A` e `dev_B`, respectivamente. Cada dispositivo tem acesso a um attestable (`att_A` e `att_B`, respectivamente). Na Figura 2, os attestables são mostrados com linhas duplas para enfatizar seu escudo de segurança. Eles estão representados como incorporados aos dispositivos, mas podem estar em outro lugar, por exemplo, na nuvem. Alice e Bob iniciam e conduzem a execução do protocolo a partir de suas respectivas aplicações (`app_A` e `app_B`). Presume-se que os *attestables* possuam recursos criptográficos para construir canais seguros para se comunicar com o PBB e entre si. Utilizamos um cenário simples para explicar as ideias principais, no qual Alice e Bob realizam a operação *handshake offline*. Utilizamos a seguinte notação:

- `D_A`: item de Alice,
- `D_B`: item de Bob,

- [D_A]: item criptografado de Alice,
- [D_B]: item criptografado de Bob,
- att_A: attestable de Alice,
- att_B: attestable de Bob,
- app_A: aplicação de Alice,
- app_B: aplicação de Bob,
- Sync_A e Cancel_A: tokens de Alice,
- Sync_B e Cancel_B: tokens de Bob.

As quatro operações principais são descritas a seguir:

1. **Deposit:** Alice e Bob depositam versões criptografadas de seus itens nos attestables um do outro. Como resultado, após a descriptografia, D_A fica bloqueado em att_B e D_B fica bloqueado em att_A.
2. **Verify:** att_A verifica as propriedades de D_B. Do lado de Bob, att_B verifica as propriedades de D_A. Uma das partes que não estiver satisfeita com o resultado da verificação ou abandona silenciosamente ou cancela o protocolo.
3. **Synchronise:** att_A envia um token Sync_A ao PBB para expressar seu interesse em continuar o protocolo. De forma semelhante e independente, att_B envia Sync_B. Para conhecer o resultado, att_A e att_B recuperam independentemente o log de tokens do PBB.
4. **Release:** Se att_A recuperar Sync_A e Sync_B do log, ele libera D_B para app_A. Consequentemente, do lado de Bob, att_B também recuperará Sync_A e Sync_B e liberará D_A para app_B. No FEWD, a execução de *restore* nunca é necessária. Nenhuma operação é executada quando o protocolo é cancelado.

Em resumo, os itens são liberados através da execução da operação release para as aplicações de seus novos proprietários (app_A e app_B) somente quando Alice publica Sync_A e Bob publica Sync_B. Itens não liberados permanecem bloqueados indefinidamente dentro dos attestables. Como explicado em [Molina-Jimenez et al. 2024], operações restore são necessárias apenas na troca de algumas classes de itens; este tópico não é abordado neste artigo. Os itens permanecem bloqueados quando Alice, Bob ou ambos cancelam a troca.

Como mostrado na Figura 2, do ponto de vista de sua funcionalidade, o FEWD é dividido em duas partes principais: manipulação de documentos e sincronização. Na manipulação de documentos, operações como *deposit*, *verify* e *release/restore* de itens são realizadas e coordenadas pelos dispositivos e seus ambientes confiáveis. Esta parte mantém o estado do protocolo e manipula informações sensíveis. Na parte de sincronização, tokens são gerados e trocados para orientar o resultado da transação entre as partes envolvidas. Esta última é utilizada por Alice e Bob exclusivamente para gerar um resultado binário (1 ou 0), que no FEWD é interpretado como liberação ou bloqueio de ambos os itens, respectivamente. A sutileza é que esse resultado pode ser produzido sem qualquer conhecimento das demais partes do protocolo. Como demonstrado na Seção 7, a operação de sincronização separada permite preservar a privacidade e outras propriedades.

O exemplo na Figura 3 ilustra os dois possíveis resultados do protocolo FEWD: (a) `Exchange Successful` e (b) `Exchange Canceled`. Em (a), o protocolo executa a seguinte sequência: (1) – o *attestable* de Alice (`att_A`) publica um *token* `Sync_A` no PBB; (2) – o *attestable* de Bob (`att_B`) publica um *token* `Sync_B` no PBB; (3) – o *attestable* de Alice recupera ambos os *tokens* `Sync_A` e `Sync_B` do PBB; (4) – o *attestable* de Bob também recupera ambos os *tokens* `Sync_A` e `Sync_B` do PBB. Como resultado, ambas as partes confirmam o compromisso mútuo de prosseguir e o protocolo alcança o estado `Exchange Successful`.

Em (b), o protocolo executa a seguinte sequência: (1) – o *attestable* de Alice inicia a troca publicando `Sync_A` no PBB na esperança de concluir a troca com sucesso; (2) – em contraste, o *attestable* de Bob decide cancelar o protocolo e publica `Cancel_B`; (3) – o *attestable* de Alice recupera ambos os *tokens* `Sync_A` e `Cancel_B` do PBB; (4) – o *attestable* de Bob também recupera ambos os *tokens* `Sync_A` e `Cancel_B` do PBB. Como resultado, o protocolo alcança o estado `Exchange Canceled`.

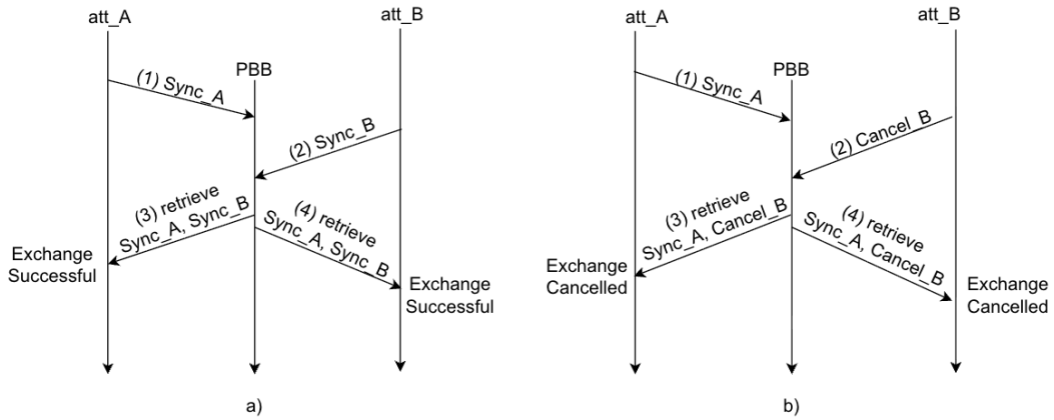

Figura 3. Resultados do FEWD: (a) Exchange successful; (b) Exchange Canceled por Bob.

3. Implicações para Sistemas de Informação

Esta seção discute as implicações organizacionais e sociotécnicas do protocolo proposto, respondendo às questões QP2 e QP3, e presentamos exemplos ilustrativos com o objetivo de facilitar a compreensão do leitor.

Em Sistemas de Informação (SI), trocas digitais não representam apenas operações técnicas, mas mecanismos estruturantes de governança, interoperabilidade e responsabilização em ecossistemas organizacionais [Jarke et al. 2019]. Em contextos interorganizacionais, a circulação de ativos digitais – como contratos, dados estratégicos e evidências eletrônicas – exige garantias explícitas de justiça, temporalidade e privacidade. A ausência dessas garantias pode gerar assimetrias informacionais e comprometer processos decisórios críticos [Markus and Silver 2008].

Essa abordagem dialoga com perspectivas sociotécnicas como a *Swift Trust Theory*, ao sustentar confiança em interações digitais sem histórico prévio, e com a *Socio-*

technical Systems Theory, ao integrar artefatos tecnológicos e estruturas organizacionais na construção de relações confiáveis.

Exemplo 1: Saúde Pública

Um exemplo concreto de aplicação do protocolo pode ser observado em um cenário emergencial de saúde pública, no qual o Hospital "A" necessita trocar rapidamente arquivos clínicos com o Hospital "B" para cruzar dados sobre a propagação de uma epidemia. Os arquivos contêm informações pessoais sensíveis de pacientes – como identificação, histórico médico e resultados laboratoriais – protegidas por regulamentações de privacidade. Nesse contexto, não há tempo hábil para estabelecer previamente um terceiro totalmente confiável (TTP) ou negociar mecanismos institucionais complexos de mediação. Ainda assim, ambas as instituições precisam garantir que a troca ocorra de forma justa: o Hospital "A" só deve liberar seus dados se o Hospital "B" também disponibilizar os seus, e vice-versa.

Exemplo 2: Cadeias de Suprimento Digitais

Outro exemplo relevante pode ser observado em cadeias de suprimento digitais, nas quais a Empresa "A" precisa trocar especificações técnicas confidenciais com o Fornecedor "B" para viabilizar a produção urgente de componentes críticos. Os documentos incluem projetos proprietários, contratos e dados estratégicos cuja exposição unilateral poderia gerar perdas competitivas significativas. Como as partes podem estar localizadas em diferentes jurisdições e não compartilhar uma autoridade central comum, a utilização de um TTP tradicional pode ser inviável ou indesejável. O protocolo de sincronização garante que os arquivos sejam liberados apenas mediante cumprimento mútuo das condições acordadas, assegurando justiça, rastreabilidade e proteção contra comportamento oportunista.

Assim, o protocolo pode ser interpretado como um mecanismo operacional de interoperabilidade sociotécnica, fornecendo suporte técnico à governança digital distribuída.

4. Metodologia

A especificação do FEWD foi publicada em 2024 [Molina-Jimenez et al. 2024] e até este momento, não há relatos de implementação. O protocolo é composto por duas partes distintas (ver Seção 2.5), que podem ser implementadas e verificadas de forma independente. Neste artigo, tanto a modelagem formal quanto a implementação foram realizadas para a parte de sincronização da troca, ou seja, o protocolo de sincronização. Esse protocolo opera por meio da postagem e recuperação de tokens em um PBB, que se restringe a receber, armazenar e disponibilizar logs quando solicitado.

A representação formal de um protocolo de sincronização permite a análise de suas propriedades gerais, tais como: ausência de *deadlocks*, alcançabilidade de estados válidos; e conformidade com regras de interação. Para oferecer uma representação rigorosa dos possíveis estados e transições do protocolo de sincronização, as FSMs das duas partes envolvidas na troca foram modeladas e formalmente especificadas. Essa abordagem fornece uma visão estruturada do comportamento do protocolo, permitindo realizar,

por meio das FSMs, a análise e a verificação das propriedades gerais e específicas definidas.

As FSMs permitem uma análise visual do protocolo de sincronização, possibilitando observar as transições entre estados e compreender o fluxo de execução, sem recorrer inicialmente à verificação formal. Para complementar essa análise e realizar a verificação automática das propriedades formais, foi utilizado o verificador *Promela/Spin* em conjunto com fórmulas expressas em *Linear Temporal Logic - (LTL)*. Essa combinação metodológica permitiu analisar formalmente as propriedades específicas do protocolo, como *strong fairness*, *strong timeliness* e privacidade.

Para responder às perguntas de pesquisa propostas, inicialmente, as FSMs foram traduzidas para a linguagem *Promela*, permitindo que o comportamento especificado do protocolo fosse analisado no ambiente do verificador *SPIN*. O processo de verificação foi então conduzido em duas etapas complementares, seguindo a metodologia padrão de análise adotada por essa ferramenta.

Na primeira etapa, o modelo foi executado sem a inclusão de fórmulas em LTL, de modo que o *SPIN* verificasse automaticamente as propriedades gerais que devem ser satisfeitas em qualquer protocolo. Entre essas propriedades estão a ausência de *deadlocks* e ciclos infinitos, garantindo que todos os caminhos de execução sejam finitos e levem a um término definido. Como essas propriedades foram satisfeitas, a segunda etapa concentrou-se na verificação das propriedades específicas do protocolo.

Na segunda etapa, incluímos fórmulas LTL no código *Promela* para que o verificador *SPIN* gerasse todos os caminhos possíveis que podem ser seguidos durante a execução do protocolo para chegar do estado inicial, a um dos estados finais (sucesso ou cancelamento). O *SPIN* gera um arquivo *.trail* por cada caminho. Em nossos códigos o *SPIN* gerou cerca de 500 arquivos *.trail*, por meio dos quais foi possível analisar propriedades específicas do protocolo de sincronização. Embora o *SPIN* não verifique automaticamente propriedades como *strong fairness*, *strong timeliness* e privacidade (essas dependem do comportamento individual de cada aplicação), foram definidas fórmulas em LTL (veja os Algoritmos 1 e 2) com o objetivo de produzir as execuções que levam aos casos de `Sucesso` e de `Cancelamento`.

Com base nos arquivos *.trail* obtidos, foi possível examinar o comportamento do protocolo em diferentes cenários e confirmar condições que expressam as propriedades esperadas. Esses arquivos permitiram, por exemplo, verificar que, sempre que Alice alcança o estado de sucesso, Bob também o alcança, garantindo a propriedade de *strong fairness*; ou ainda que, se Bob abandona o protocolo (não envia nenhum token), Alice eventualmente pode cancelar, garantindo a propriedade de *strong timeliness*. Além disso, a análise das sequências contidas nos arquivos *.trail* possibilitou comprovar a garantia de privacidade, uma vez que tanto Alice quanto Bob atingem seus estados finais de sucesso ou de cancelamento, publicando no PBB apenas tokens do vocabulário definido (`Sync_A`, `Sync_B`, `Cancel_A` e `Cancel_B`), os quais consistem apenas em cadeias de caracteres sem qualquer informação sensível.

Por fim, após a conclusão da etapa de *model checking*, o protocolo de

sincronização foi implementado. A implementação foi submetida à verificação formal das propriedades especificadas no modelo. Todas as sequências geradas pelo verificador *SPIN* foram testadas na implementação para verificar sua conformidade. Essa etapa prática teve como objetivo verificar, em um ambiente de execução real, os comportamentos e garantias identificados durante a modelagem e a verificação formal. A integração entre verificação e implementação reforça a consistência entre o modelo teórico e o comportamento observado na prática, evidenciando a correspondência entre a especificação formal e a execução efetiva do protocolo.

5. Modelagem e Verificação do Protocolo de Sincronização

Esta seção aborda a QP4, que investiga como validar formalmente que as propriedades reivindicadas, em especial *strong fairness*, cancelamento unilateral imediato e preservação da privacidade, são efetivamente satisfeitas pelo protocolo proposto. Para isso, adotou-se uma abordagem baseada em modelagem formal e verificação por *model checking*, permitindo analisar sistematicamente o comportamento do protocolo sob diferentes cenários de execução.

O protocolo foi modelado como uma FSM. Como o protocolo é simétrico, optou-se por apresentar apenas a FSM de Alice (ver Figura 4). A FSM de Bob é estruturalmente equivalente, com estados e transições correspondentes, diferenciando-se apenas na identidade do participante e pelos tokens utilizados. Nessa modelagem, cada transição representa uma ação válida do protocolo, como o envio dos tokens `Syn_A` ou `Cancel_A`, refletindo as decisões possíveis de Alice ao longo da execução. A FSM de Bob segue a mesma lógica, porém utilizando os tokens `Sync_B` e `Cancel_B`. O protocolo possui dois estados finais válidos: "*Protocol Success*", quando a troca é concluída com êxito; e "*Protocol Canceled*", quando a execução é encerrada de forma segura por qualquer um dos participantes. A partir deste modelo, o protocolo foi especificado na linguagem *Promela*, e fórmulas em LTL foram empregadas, possibilitando a verificação formal apresentada na Subseção 5.1.

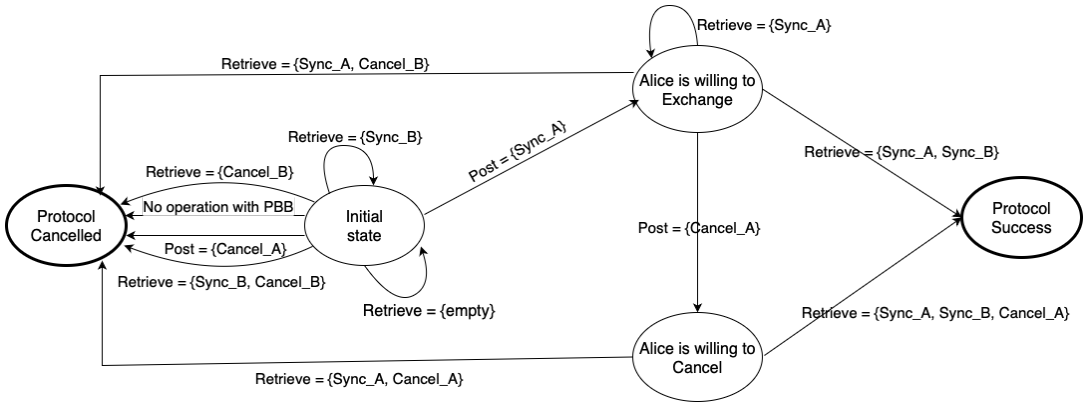

Figura 4. FSM de Alice

As fórmulas em LTL apresentadas nos Algoritmos 1 e 2 foram incluídas no modelo *Promela* para gerar sequências de execução específicas. O código completo pode ser visto em: `https://github.com/gca-research-group/fair-exchange/`

`tree/main/model-checking`. No Algoritmo 1, a fórmula `<> (ali_succ && bob_succ)` garante que o *SPIN* explore todas as trajetórias em que tanto Alice quanto Bob terminam em seus respectivos estados de `Sucesso`, produzindo arquivos .trail correspondentes a essas execuções. Como ilustrado na Figura 5, em um dos arquivos *.trail*, os estados são representados pelas variáveis `ali_succ`, `ali_canc`, `bob_succ` e `bob_canc`. As variáveis `ali_succ` e `bob_succ` assumem o valor `1` quando os participantes Alice e Bob alcançam o estado de `Sucesso`, enquanto `ali_canc` e `bob_canc` assumem o valor `0` quando eles atingem o estado de `Cancelamento`. Dessa forma, essas variáveis funcionam como indicadores binários que representam o progresso de cada participante durante a execução do protocolo.

Algorithm 1 Fórmula em LTL que gera os arquivos .trail onde Alice e Bob terminam em Sucesso

```
never  {    /* <>(ali_succ && bob_succ) */
T0_init:
        do
        :: atomic { ((ali_succ && bob_succ)) -> assert(!((ali_succ && bob_succ))) }
        :: (1) -> goto T0_init
        od;
accept_all:
        skip
}
```

No Algoritmo 2, a fórmula `[] !(ali_succ && bob_succ)` instrui o *SPIN* a percorrer todas as sequências em que tanto Alice quanto Bob terminam em seus respectivos estados de `Cancelamento`.

Algorithm 2 Fórmula em LTL que gera arquivos .trail onde Alice ou Bob terminam em estado de cancelamento

```
never  {    /* []!(ali_succ && bob_succ) */
accept_init:
T0_init:
        do
        :: (! ((ali_succ && bob_succ))) -> goto T0_init
        od;
}
```

Essas fórmulas em LTL orientam o *SPIN* na exploração sistemática de todas as possíveis execuções do modelo, permitindo a identificação precisa de comportamentos desejáveis e indesejáveis. Neste trabalho, os arquivos gerados a partir dessa verificação permitem realizar uma análise formal das propriedades específicas do protocolo FEWD. O código utilizado para a geração dos arquivos *.trail* está disponível em: `https://github.com/gca-research-group/fair-exchange/tree/main/model-checking`.

5.1. Análise das Propriedades Strong Fairness e Strong Timeliness

A Figura 5 apresenta um trecho de um arquivo *.trail* gerado pelo verificador *SPIN*, contendo o estado final válido de cada processo no modelo, bem como os valores das variáveis globais e locais.

```
                pbb_succ = 0
                pbb_canc = 1
                ali_succ = 0
                ali_canc = 1
                bob_succ = 0
                bob_canc = 1
                pbbLog[0] = sync_a
                pbbLog[1] = cancel_a
                pbbLog[2] = sync_b
                pbbLog[3] = 0
                pbbLog[4] = 0
                pbbLog[5] = 0
                aliLog[0] = sync_a
                aliLog[1] = cancel_a
                aliLog[2] = sync_b
                aliLog[3] = 0
                aliLog[4] = 0
                aliLog[5] = 0
                bobLog[0] = sync_a
                bobLog[1] = cancel_a
                bobLog[2] = sync_b
                bobLog[3] = 0
                bobLog[4] = 0
                bobLog[5] = 0
                pbbLogIndex = 3
393:    proc  3 (alice:1) pbb_ali_bob.pml:539 (state 64) <valid end state>
393:    proc  2 (bob:1) pbb_ali_bob.pml:630 (state 63) <valid end state>
393:    proc  1 (fsm_of_alice:1) pbb_ali_bob.pml:445 (state 224)
393:    proc  0 (:init::1) pbb_ali_bob.pml:642 (state 4) <valid end state>
393:    proc  - (never_0:1) pbb_ali_bob.pml:683 (state 3)
```

Figura 5. Arquivo .trail gerado pelo verificador SPIN

Os resultados da verificação formal confirmaram que o modelo está livre de *deadlocks* e de ciclos infinitos, garantindo que todas as execuções terminem corretamente. Todas as trajetórias analisadas convergiram para um dos dois estados finais válidos: "*Protocol Success*" ou "*Protocol Canceled*", o que demonstra a completude e a correção do comportamento especificado. Além disso, as propriedades específicas definidas para o protocolo foram verificadas com sucesso, reforçando a consistência entre a modelagem baseada em FSMs, a especificação formal em *Promela* e o processo de verificação realizado com *SPIN*.

A seguir, realiza-se a análise detalhada das propriedades específicas do protocolo, com base nas FSMs e nos arquivos *.trail* gerados pelo verificador *SPIN*.

5.1.1. Strong Fairness

Para evidenciar a propriedade de *strong fairness*, observa-se que, na operação de *retrieve*, Alice e Bob acessam o mesmo log no PBB e alcançam, de forma consistente, um estado final idêntico. Essa característica é demonstrada nos arquivos *.trail* gerados pelo *SPIN*, nos quais as variáveis `ali_succ`, `ali_canc`, `bob_succ` e `bob_canc` assumem valores binários ("1" ou "0"). Quando `ali_succ` e `bob_succ` recebem o valor "1", ambos os participantes atingem o estado de *Protocol Success* de forma sincronizada. Da mesma maneira, quando `ali_canc` e `bob_canc` assumem o valor "1", o protocolo é encerrado no estado de *Protocol Canceled* para ambas as partes, evidenciando a consistência e a sincronização entre os estados finais.

A seguir, são apresentadas algumas das sequências geradas pelo *SPIN* e as respectivas FSMs, ilustrando o percurso correspondente a cada sequência.

Estado Final (Protocol Success): Na Figura 6, observa-se que a variável `ali_succ` assume o valor "1" e `ali_canc` recebe o valor "0", indicando que o protocolo atingiu o estado "*Protocol Success*" para Alice. De forma análoga, no lado de Bob, a variável `bob_succ` assume o valor "1" e `bob_canc` recebe o valor "0", indicando que Bob também atingiu o estado de "*Protocol Success*". No arquivo *.trail*, observa-se que o log `sync_a`, `sync_b` , armazenado no PBB após a operação de *retrieve*, é idêntico para Alice e Bob, evidenciando que o protocolo foi concluído em um estado consistente e justo para ambas as partes. Quando a operação *synchronise* é concluída no estado de "*Protocol Success*", os *attestables* executam a operação *release*, liberando os itens para seus respectivos donos.

Figura 6. Saída do Spin (*Strong Fairness*) - Alice e Bob ("*Protocol Success*")

Analisando a sequência gerada pelo *SPIN*, é possível relacioná-la à Figura 4 e reconstruir o percurso realizado por Alice. Inicialmente, ela envia seu token `Sync_A`, indicando a intenção de prosseguir com a troca. Em seguida, ao executar a operação de *retrieve*, verifica que o log do PBB contém tanto o token `Sync_A` quanto o token `Sync_B`, publicado por Bob. Esse registro confirma que ambas as partes manifestaram intenção de concluir a troca.

Dessa forma, a FSM de Alice transita do estado "*Initial state*" para o estado textcolorblue"*Alice is willing to Exchange*", até alcançar o estado final "*Protocol Success*". De maneira simétrica, a FSM de Bob percorre a mesma trajetória. Esse comportamento é evidenciado nos resultados do *SPIN*, em que as variáveis `ali_succ = 1` e `bob_succ = 1` indicam a conclusão bem-sucedida do protocolo por ambas as partes.

Estado Final (Protocol Canceled): Na Figura 7, observa-se que a variável `ali_succ` assume o valor "0" e `ali_canc` recebe o valor "1", indicando que o protocolo atingiu o estado "*Protocol Canceled*" do lado de Alice. De forma análoga, as variáveis `bob_succ` e `bob_canc` assumem, respectivamente, os valores "0" e "1", evidenciando que Bob também alcançou o estado de "*Protocol Canceled*". Nesse cenário, os itens permanecem bloqueados nos *attestables*. Esse resultado, obtido por meio da verificação no *SPIN*, demonstra a propriedade de *strong fairness*, uma vez que ambos os participantes convergem para o mesmo estado final do protocolo.

Com base em uma das sequências geradas pelo *SPIN* para o cenário de cancelamento, observa-se que Alice inicialmente publica o token `Sync_A`, manifestando sua intenção de prosseguir com a troca. Ao executar a operação *retrieve*, verifica que o log do PBB ainda não contém o token de Bob. Persistindo, publica novamente `Sync_A` e realiza um novo *retrieve*. Nessa segunda consulta, o log retorna o conjunto `Sync_A`, `Sync_A`, `Cancel_B` , indicando que Bob optou por encerrar a troca. Neste caso, Alice transita do estado *Initial state*" para *Alice is willing to Exchange*" e, em seguida, para o estado final *Protocol Canceled*". De forma correspondente, Bob apenas publica o token `Cancel_B` e encerra a execução no estado *Protocol Canceled*". Essa trajetória demonstra que o protocolo trata corretamente a desistência de uma das partes, assegurando a terminação segura

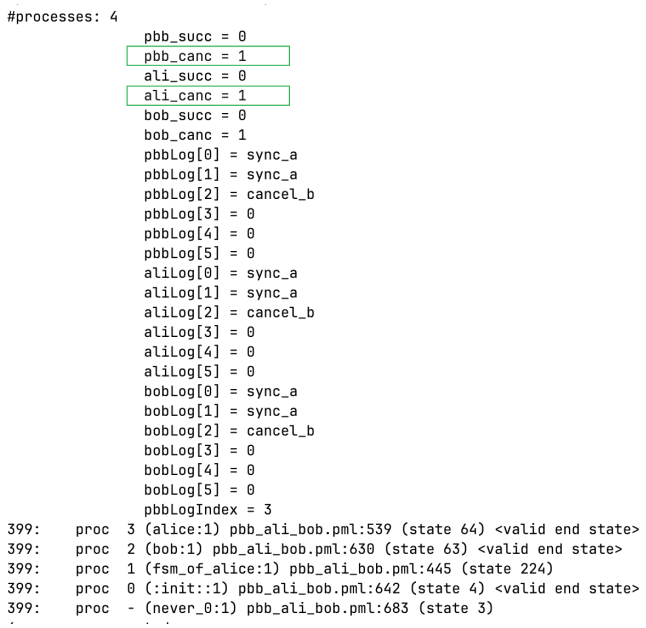

Figura 7. Saída do Spin *Strong Fairness* - Alice e Bob ("*Protocol Canceled*")

e justa da execução. Esse percurso pode ser claramente reconstruído a partir da Figura 4, que ilustra as transições correspondentes na FSM de Alice.

A análise confirma que as FSMs foram modeladas corretamente, pois todas as execuções convergem para um dos dois estados finais válidos: "*Protocol Success*" ou "*Protocol Canceled*". Foram testadas inúmeras sequências geradas pelo *SPIN*, e, ao serem representadas nas FSMs, todas terminavam consistentemente em um dos dois estados finais possíveis. Os arquivos *.trail* gerados pelo *SPIN* confirmam que o protocolo satisfaz a propriedade de *strong fairness*. Ao executarem a operação *retrieve*, tanto Alice quanto Bob acessam o mesmo log do PBB e convergem para o mesmo resultado, seja sucesso ("*Protocol Success*") ou cancelamento ("*Protocol Success*"). Cada arquivo *.trail* contém uma sequência que sempre conduz a um dos estados finais, condição indicada pelas variáveis `ali_succ`, `ali_canc`, `bob_succ` e `bob_canc`, que assumem ao longo dos arquivos *.trail* os valores "1" ou "0".

5.1.2. Strong Timeliness

São analisados a seguir cenários em que a propriedade de *strong timeliness* é satisfeita. Essa propriedade assegura que os participantes mantenham controle sobre a execução do protocolo, permitindo cancelar a troca antes de atingir um estado final, seja porque Alice mudou de ideia, seja porque Bob desapareceu.

A Figura 8 apresenta uma das sequências obtidas a partir do modelo verificado pelo *SPIN*. O arquivo *.trail* analisado contém a sequência { `Sync_A`, `Sync_A`, `Cancel_A` }, representando um cenário em que Alice realiza duas tentativas consecutivas de sincronização, enviando duas vezes o token `Sync_A`. Diante da ausência de resposta de

Bob, Alice decide encerrar a execução do protocolo, enviando o token de cancelamento `Cancel_A`. Essa sequência conduz o protocolo ao estado final "*Protocol Canceled*", o que é confirmado pelos valores das variáveis `ali_canc` e `bob_canc`, ambos assumindo o valor "1".

```
#processes: 4
                pbb_succ = 0
                pbb_canc = 1
                ali_succ = 0
                ali_canc = 1
                bob_succ = 0
                bob_canc = 1
                pbbLog[0] = sync_a
                pbbLog[1] = sync_a
                pbbLog[2] = cancel_a
                pbbLog[3] = 0
                pbbLog[4] = 0
                pbbLog[5] = 0
                aliLog[0] = sync_a
                aliLog[1] = sync_a
                aliLog[2] = cancel_a
                aliLog[3] = 0
                aliLog[4] = 0
                aliLog[5] = 0
                bobLog[0] = sync_a
                bobLog[1] = sync_a
                bobLog[2] = cancel_a
                bobLog[3] = 0
                bobLog[4] = 0
                bobLog[5] = 0
                pbbLogIndex = 3
399:    proc  3 (alice:1) pbb_ali_bob.pml:539 (state 64) <valid end state>
399:    proc  2 (bob:1) pbb_ali_bob.pml:630 (state 63) <valid end state>
399:    proc  1 (fsm_of_alice:1) pbb_ali_bob.pml:445 (state 224)
399:    proc  0 (:init::1) pbb_ali_bob.pml:642 (state 4) <valid end state>
399:    proc  - (never_0:1) pbb_ali_bob.pml:683 (state 3)
```

Figura 8. Saída do Spin *Strong Timeliness* - Alice e Bob ("*Protocol Canceled*")

As Figuras 9 e 10 apresentam os caminhos tracejados que representam a sequência descrita no arquivo *.trail* mostrado na Figura 8. Alice inicia o protocolo publicando o token `Sync_A`, conforme ilustrado na Figura 9, para indicar sua intenção de seguir com a troca. Em seguida, ela realiza uma operação de *retrieve* e verifica que o log do PBB contém apenas seu próprio token. Diante da ausência de resposta de Bob, Alice publica um segundo `Sync_A` e realiza uma nova consulta no PBB, obtendo o log `Sync_A`, `Sync_A` , o que confirma que Bob ainda não enviou nenhum token. Diante dessa situação, Alice opta por cancelar a operação, enviando o token `Cancel_A`. O *retrieve* subsequente retorna o log `Sync_A`, `Sync_A`, `Cancel_A` , refletindo exatamente a sequência capturada pelo arquivo *.trail* gerado pelo verificador *SPIN*. Essa trajetória, representada pelas transições tracejadas na FSM, conduz o sistema do estado "*Initial state*" para "*Alice is willing to Exchange*", em seguida para "*Alice is willing to Cancel*", e finalmente para o estado terminal "*Protocol Canceled*". O resultado confirma que Alice pode encerrar o protocolo sem bloqueio, assegurando a propriedade de *strong timeliness*. Além disso, as variáveis `ali_canc` e `bob_canc` assumem o valor "1", indicando que ambos os participantes finalizam o processo de forma consistente no mesmo estado. Na FSM de Bob, apresentada na Figura 10, observa-se que ocorreu uma demora, falha ou interrupção momentânea que o impediu de dar continuidade à troca. Ao retomar, Bob executou a operação *retrieve* e verificou que Alice já havia cancelado a troca, o que o levou diretamente ao estado de "*Protocol Canceled*".

A propriedade de *strong timeliness* demonstrada nesta seção confirma que, mesmo

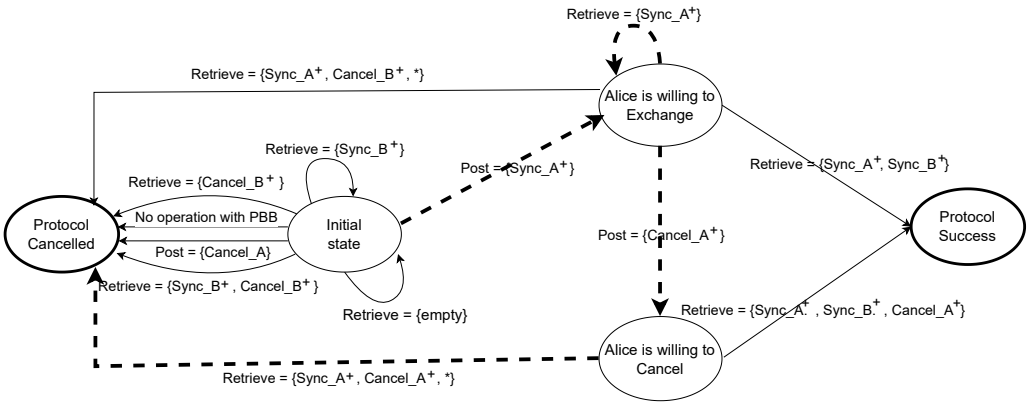

Figura 9. FSM da Alice com cancelamento - " *Timeliness*"

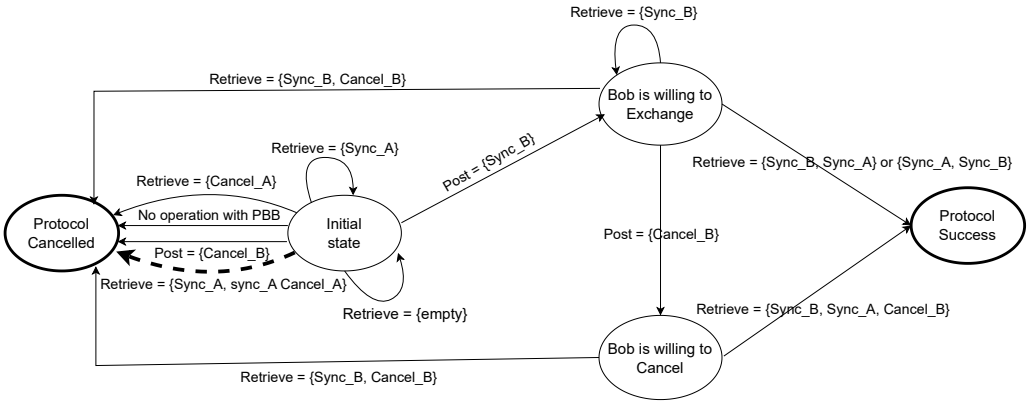

Figura 10. FSM de Bob com cancelamento - " *Timeliness*"

na ausência de ação de uma das partes, o protocolo não entra em bloqueio ou espera indefinida, garantindo sempre a conclusão em um estado final válido e justo. A análise das FSMs e das saídas do verificador *SPIN* evidencia que o protocolo de sincronização foi corretamente modelado e especificado em *Promela*. Em todas as sequências de execução verificadas, observa-se a existência de um caminho finito que leva a um estado final – seja de *Sucesso* ou de *Cancelamento* – assegurando que cada participante possa encerrar sua execução de forma imediata e independente, sem depender da ação do outro.

6. Implementação do FEWD

O protocolo de sincronização foi implementado em *Python*, seguindo o modelo discutido na Seção 5. Nesta implementação, o protocolo é iniciado pela operação de *handshake*, responsável por definir as configurações iniciais e os acordos entre as partes, apresentada na Subseção 6.1. Logo, a operação de sincronização, que está descrita na Subseção 6.2, foi implementada utilizando um PBB para armazenar e liberar *tokens* envolvidos na troca quando solicitados. Assume-se que a implementação das operações *deposit*, *verify* e *release/restore*, pertencentes ao processo de manipulação de documentos, está fora do escopo deste artigo.

6.1. Implementação da Operação Handshake e das Configurações Iniciais

Para iniciar o protocolo, Alice e Bob executam uma operação *handshake* para criar um documento de configuração que eles concordam em observar durante a execução do protocolo. Para isso, os *attestables* geram suas chaves públicas e definem as identidades dos usuários. O documento de configuração estipula que Alice e Bob concordam com os termos da troca. Embora o documento de configuração possa incluir vários parâmetros, nossa implementação inclui apenas três: **PBB** – Especifica a Uniform Resource Locator (URL) e o certificado de *Transport Layer Security* (TLS) [Dierks and Rescorla 2008] do PBB que Alice e Bob concordam em usar; **Chaves do dispositivo** – Especifica as chaves e certificados a serem usados para a *attestation* dos componentes confiáveis de Alice e Bob; **Documento** Descreve os itens que Alice e Bob estão trocando. Contém três subparâmetros: **setup_values**, **module** e **certificate**. O module contém quatro subparâmetros: **setup_values_types**, **doc_values_types**, **verify_function** e **gather_function**. Entre eles, a **gather_function** é responsável por gerar os **doc_values**.

Vamos examinar a descrição do item de Alice. A descrição é aplicada simetricamente a Bob. Por exemplo, `M_A` refere-se ao módulo na descrição do item de Alice, e `M_B` refere-se ao módulo na descrição de Bob.

Os `setup_values`, denotados por `P_A`, são os valores fornecidos por Alice. Seus tipos correspondentes formam a lista `setup_values_types`, representada como `S_A`. Os `doc_values_types`, denotados `T_A`, referem-se aos tipos esperados por Bob. A `gather_function` coleta os valores do item original de Alice. Utilizamos a notação `G_A`. Os valores coletados são colocados em uma lista de `doc_types`, ou seja, $X_A = \{x_1, \ldots, x_M\}$. Se os tipos dos valores em `X_A` coincidirem com os tipos listados em `doc_values_types`, a `gather_function` retorna `X_A`, caso contrário, retorna um erro, que cancela a troca. A `verify_function` é responsável por validar os dados. Ela recebe como entrada os `setup_values` do documento de configuração e a lista `doc_types` coletada pela `gather_function`, e produz um valor booleano: `True` se os valores em `setup_values` corresponderem aos valores em `doc_types`. Os valores `True` e `False` representam, respectivamente, a aceitação e rejeição de `D_A`.

6.2. Implementação da Operação de Sincronização

A Figura 11 apresenta a arquitetura da implementação. Os dispositivos `dev_A` e `dev_B` representam, respectivamente, os ambientes de execução de Alice e Bob. Cada dispositivo contém um módulo confiável denominado *attestable* (`att_A` e `att_B`) responsável por executar operações críticas do protocolo (*deposit*, *verify* e *release/restore*). No entanto, esses módulos não foram implementados nesta etapa do trabalho.

Os componentes `app_A Server`, `app_B Server` e `PBB Server` interagem por meio de *APIs RESTful* implementadas com o *framework Flask*. Cada entidade mantém um banco de dados independente em *PostgreSQL*, garantindo isolamento e persistência de estado. Toda a comunicação entre as entidades ocorre por meio de canais autenticados e criptografados via TLS (*TLS secchans*). Para simplificar o processo de *deployment* e assegurar portabilidade entre sistema operacional, foi utilizado o *Docker*. Foram criados três contêineres: um para o `PBB Server`, um para a aplicação da Alice (`app_A Server`) e outro para a aplicação do Bob (`app_B Server`).

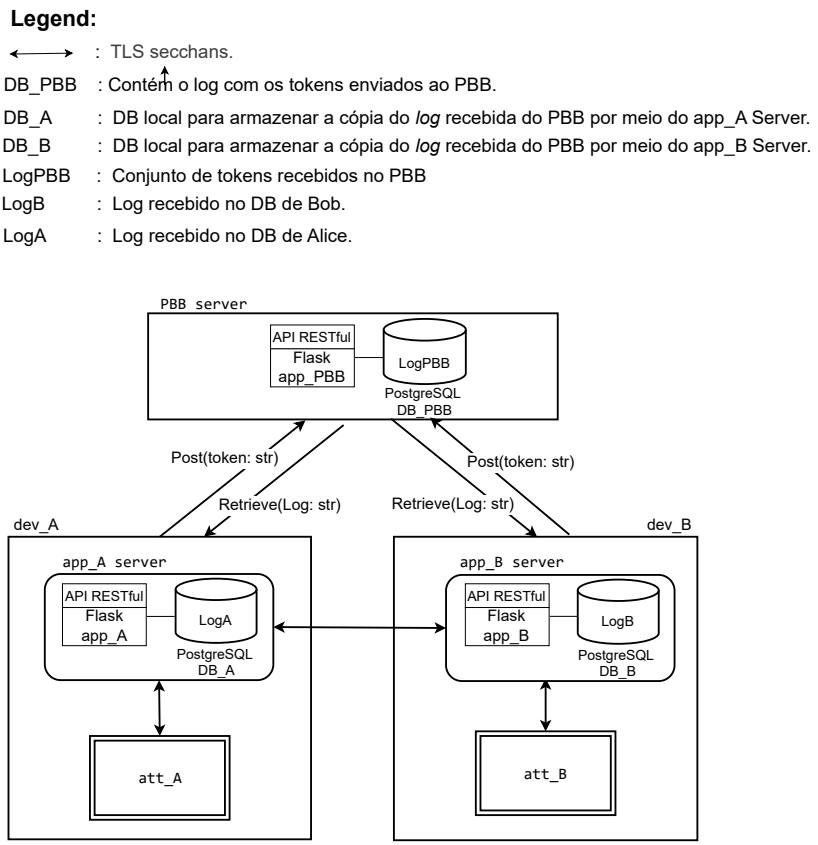

Figura 11. Arquitetura da implementação do protocolo.

O `PBB Server` atua como repositório de tokens, recebendo as mensagens `POST(token: str)` enviadas pelas aplicações `app_A Server` e `app_B Server` e registrando-as no log armazenado em seu banco de dados (`PostgreSQL DB_PBB`). Quando a aplicação `app_A` executa a operação `Retrieve(Log: str)`, o `PBB Server` retorna o log `logPBB`, que é então armazenado localmente no banco de dados `PostgreSQL DB_A` como `LogA`. O mesmo ocorre com a `app_B`, que executa a operação correspondente e armazena uma cópia local `LogB` em seu `PostgreSQL DB_B`. Ou seja, após a execução da operação *retrieve*, existirão três cópias do log: `LogPBB`, armazenada no `PostgreSQL BD_PBB`, `LogA` no `PostgreSQL BD_A` e `LogB` no `PostgreSQL BD_B`. As aplicações de Alice e Bob, enviam `LogA` e `LogB` aos respectivos *attestables* (`att_A` e `att_B`). Com base nesses logs, os *attestables* podem computar o resultado da troca (sucesso ou cancelamento) e realizar a operação *release/-restore*, liberando ou retendo os itens de acordo com o desfecho da troca.

6.2.1. Implementação do PBB Stateless

Os *attestables* de Alice e Bob por meio das suas respectivas aplicações, `app_A` e `app_B`, publicam tokens (`Sync_A`, `Cancel_A`, `Sync_B` ou `Cancel_B`) no PBB, representados

pela função `post` no Algoritmo 3.

Algorithm 3 Função post token no PBB

```
1  def post(exchange_token: str):
2      data = request.get_json()
3      base_dir = f"data/{exchange_token}"
4      os.makedirs(base_dir, exist_ok=True)
5
6      with open(f"{base_dir}/encrypted_item", "w") as item_file:
7          item_file.write(data.get("encrypted_item"))
8
9      try:
10         verify(data.get("doc_value", {}))
11     except ValueError as e:
12         return Response(str(e), status=400)
13
14     return Response(status=200)
```

A função `post(exchange_token: str)` recebe e processa dados enviados via requisição HTTP no formato JSON. Inicialmente, na linha 2, o conteúdo da requisição é obtido por meio de `request.get_json()`, sendo armazenado na variável `data`. Em seguida, é criado um diretório específico `(data/{exchange_token})` para armazenar informações associadas àquela transação, garantindo sua criação caso ainda não exista `(os.makedirs(..., exist_ok=True))`, nas linhas 3 e 4. O item criptografado recebido é salvo em arquivo local, denominado `encrypted_item`. Após o armazenamento, a função invoca o método `verify()` para validar os valores do documento `(doc_value)` enviados na requisição. Caso a verificação identifique inconsistências, é retornada uma resposta HTTP com código 400 (erro de requisição inválida). Em condições normais, a função responde com o código 200, indicando que a operação foi concluída com sucesso.

O Algoritmo 4 define a função que cria uma nova troca (`exchange`) no PBB, associando-a ao usuário que a inicia. Após validar os parâmetros e verificar a inexistência de uma troca com o mesmo `exchange token`, a função registra a nova troca e associa os tokens dos participantes ao identificador correspondente, permitindo a recuperação do log. A função retorna o código HTTP 201 `Created`, indicando o sucesso da operação.

Inicialmente, as linhas 3 a 6 obtêm os dados enviados na requisição HTTP em formato JSON, extraindo os campos `exchange_token`, `user_token` e `user_name`. Em seguida, as linhas 8 a 9 realizam uma validação para garantir que todos esses campos estejam presentes; caso contrário, é lançada uma exceção do tipo `BadRequest`. Dentro do bloco gerenciado por `scoped_connection()` (linhas 10), a função estabelece uma conexão com o banco de dados e verifica se já existe uma troca com o mesmo `exchange_token` (linhas 11 a 14). Se o token já estiver cadastrado, é lançada uma exceção para evitar duplicidade. Caso contrário, um novo objeto `Exchange` é criado (linha 15), adicionado à sessão e gravado parcialmente com `flush()` para gerar seu identificador (linha 18). Em seguida, um usuário associado à troca é criado a partir das informações fornecidas (`ExchangeUser`) e adicionado à sessão (linhas 19–20). Por fim, a transação é confirmada com `commit()` (linha 21), e a função retorna uma resposta HTTP com o status 201, indicando a criação bem-sucedida do recurso (linha 22).

Algorithm 4 Função de criação de uma nova troca e registro de tokens no PBB

```
def create():

    data = request.get_json()
    exchange_token = data.get("exchange_token")
    user_token = data.get("user_token")
    user_name = data.get("user_name")

    if not exchange_token or not user_token or not user_name:
        raise BadRequest("exchange_token,user_token,and,user_name,arerequiblue")
    with scoped_connection() as connection:
        existing_exchange = (connection.query(Exchange).filter_by
        (token=exchange_token).first()
        if existing_exchange:
            raise BadRequest("Exchange_with_this_token_already_exists")
        exchange = Exchange(token=exchange_token)

        connection.add(exchange)
        connection.flush()
        user = ExchangeUser(exchange_id=exchange.id, token=user_token, name=user_name)
        connection.add(user)
        connection.commit()
    return Response(status=201)
```

A função `retrieve(exchange_token: str)`, no Algoritmo 5, tem como objetivo recuperar o log com os tokens previamente armazenado no PBB. Para isso, o código constrói o caminho do arquivo com base no identificador único da troca `(exchange_token)` e verifica sua existência no diretório correspondente. Caso o arquivo não seja encontrado, a função retorna uma resposta HTTP com código 404, indicando ausência do recurso solicitado.

Algorithm 5 Função que recuperar (retrieve) o log de tokens do PBB

```
def retrieve(exchange_token: str):
    path = f"data/{exchange_token}/encrypted_item"
    if not os.path.exists(path):
        return Response("File_not_found", status=404)

    with open(path, "r") as item_file:
        content = item_file.read()

    return Response(content, status=200, mimetype="text/plain")
```

Quando o arquivo é localizado, seu conteúdo é liberado para quem solicitou por meio de uma resposta HTTP com código 200, sinalizando sucesso na operação. O parâmetro `mimetype="text/plain"` define o tipo de conteúdo da resposta, assegurando que o log seja entregue em formato de texto simples.

O PBB é responsável por registrar a troca criada e armazenar todos os tokens relacionados a essa troca em log. Assim, cada vez que uma nova troca é criada, os dados associados – como o identificador da troca, os tokens de usuários e as ações de aceitação – são registrados no sistema de armazenamento do PBB. Posteriormente, quando houver uma solicitação de recuperação, o PBB é capaz de enviar o log correspondente ao solicitante, permitindo o solicitante verificar o histórico completo dos tokens armazenados.

Nesta versão estendida, também realizamos uma análise mais aprofundada do comportamento do protocolo, discutindo implicações práticas para sua integração em sistemas de informação distribuídos.

7. Análise das Propriedades

Esta versão da implementação do protocolo introduz novos cenários de teste e verificação, permitindo avaliar se o protocolo de sincronização implementado preserva as propriedades *strong fairness*, *strong timeliness* e privacidade.

7.1. Strong Fairness

De acordo com [Molina-Jimenez et al. 2024], a *strong fairness* só pode ser garantida em protocolos nos quais a operação de *synchronise* é executada em um ambiente de mensagens independente, ou seja, um serviço de mensagens que não pode ser controlado pelos participantes. Em nossa implementação, esse ambiente é fornecido pelo PBB. Na Figura 12, apresentamos a FSM com os possíveis estados do protocolo e suas transições, a qual foi utilizada como base para a implementação, garantindo a propriedade de *strong fairness*. Essa FSM possui cinco estados: um inicial, dois intermediários e dois finais. Existem oito caminhos possíveis do estado inicial aos estados finais.

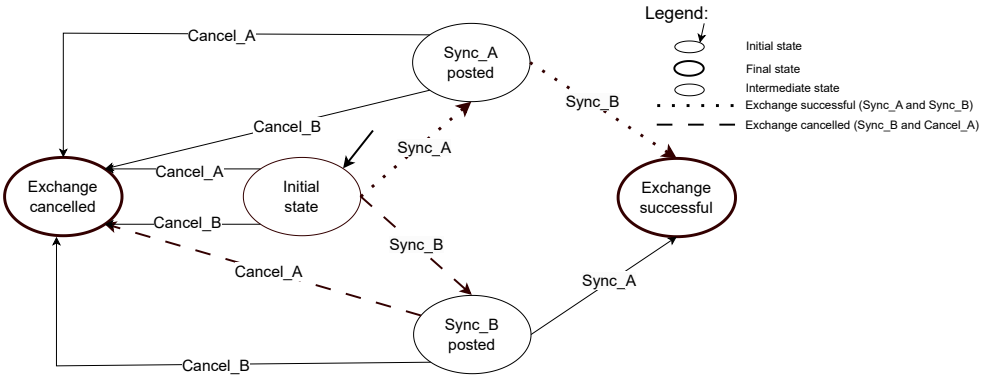

Figura 12. FSM da operação de sync com seus dois estados finais únicos e mutuamente exclusivos.

A propriedade *strong fairness* é garantida porque a FSM sempre progride do estado inicial para um dos dois estados finais mutuamente exclusivos; ou `Exchange successful` ou `Exchange Canceled`. Na Figura 12, ilustramos os caminhos que progridem do estado inicial para um dos estados finais, ou seja, `Exchange successful`. Também mostra os caminhos que progridem do estado inicial para o estado `Exchange Canceled`, isto é, o estado final onde a troca é cancelada.

A verificação da implementação foi realizada por meio dos arquivos *.trail* gerados pelo *SPIN*. Constatou-se que todas as sequências de execução produziram os mesmos resultados, tanto para Alice quanto para Bob. Os testes confirmaram que a implementação reproduz fielmente a propriedade *strong fairness* do protocolo. A execução da operação *retrieve*, ilustrada na Figura 13, evidencia de forma empírica a propriedade de *strong*

fairness. Observa-se que tanto o attestable de Alice quanto o de Bob recebem, do PBB, o mesmo log contendo uma sequência idêntica de tokens (`Sync_A`, `Sync_A`, `Sync_B`). Essa simetria confirma que o PBB atua como um ambiente de mensagens independente, responsável por receber os tokens e disponibilizar o log, sempre que requisitado. Observa-se na figura que ambos os participantes, Alice e Bob, recebem o mesmo log, o que garante que o protocolo alcance um dos estados finais de forma consistente e idêntico para ambos.

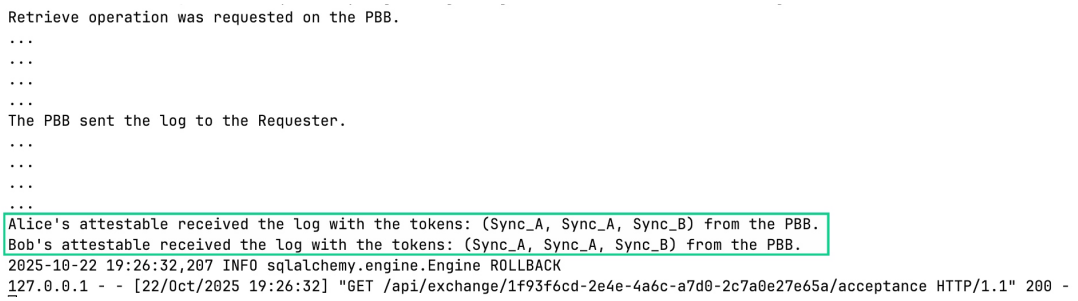

Figura 13. Strong Fairness garantida para Alice e Bob.

7.2. Privacidade

Em SFEPs, a privacidade é colocada em risco pelas informações que Alice e Bob revelam ao TTP. Como explicado na Seção 2, o TTP é necessário pelo menos para a execução da operação de *synchronise* [Pagnia and Darmstadt 1999]. A Figura 12 mostra como nossa implementação pode executar a operação de *synchronise* sem revelar informações sensíveis ao PBB: observe que a FSM utiliza apenas quatro strings independentes de aplicação como tokens: `Sync_A`, `Cancel_A`, `Sync_B` e `Cancel_B` para sincronizar em `Exchange Canceled` ou `Exchange successful`. Na prática, as strings podem incluir algumas informações significativas para Alice e Bob (por exemplo, chaves e assinaturas), mas não precisam incluir informações sobre os itens em troca, as identidades das partes envolvidas ou o desenvolvimento da troca. Portanto, não precisam ser ocultadas do PBB ou de outras partes que possam ter acesso a ele. Com essa abordagem, o PBB não é apenas sem estado (stateless), mas também independente de aplicação (*application-agnostic*). Ele atua apenas como um repositório de tokens, oferecendo dois serviços principais: armazenar e entregar tokens.

Esses tokens são simplesmente cadeias de caracteres sem qualquer informação sensível, garantindo a privacidade. A figura 14 ilustra que o PBB recebeu os tokens `Sync_A`, `Sync_A` e `Sync_B`, os quais foram registrados em um log. Quando solicitado, esse log é então enviado ao participante que realizou a operação de *retrieve*.

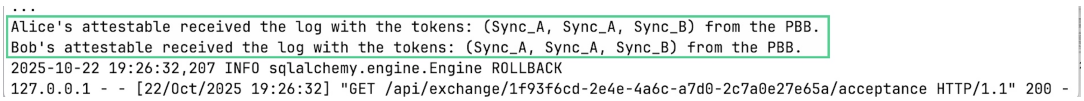

Figura 14. Tokens de caracteres enviados ao PBB durante a operação de sincronização.

Na descrição do FEWD em [Molina-Jimenez et al. 2024], o PBB aceita um token de qualquer pessoa e o adiciona ao final de um log arbitrariamente grande. Mediante

solicitação, ele entrega o log completo a qualquer pessoa. Este é certamente um modelo geral de PBB que pode ser usado para sincronização com garantias de privacidade. No entanto, a implementação de tal PBB é complexa devido à sua generalidade; além disso, o fato de que o solicitante do token sempre recupera o log completo (possivelmente com milhares de tokens) para verificar se os dois tokens de que precisa já estão ou não no log questiona a eficiência do protocolo.

Nesta implementação, simplificamos o log do PBB, assumindo que Alice e Bob executam apenas uma instância do protocolo. O PBB é responsável por receber e registrar os *tokens* em um banco de dados associado a um ID de troca único entre os participantes, garantindo que apenas os *tokens* correspondentes a essa troca específica sejam armazenados; portanto, os *tokens* precisam incluir o ID da troca. Esse ID da troca é gerado durante o *handshake* inicial entre os participantes. A associação dos *tokens* às instâncias do protocolo simplifica a filtragem de *tokens* no PBB e, consequentemente, simplifica o código que implementa o PBB. Isso também libera os *attestables* da tarefa de filtragem de *tokens*.

Ressalta-se que, no protocolo proposto, as operações de *deposit*, *verify* e *release/restore*, que contém informações sensíveis, são executadas dentro dos *attestables*, enquanto a operação de *synchronise*, que não tem informações sensíveis, é executada no PBB. O uso dos attestables para executar operações sensíveis relacionadas a documentos, sem expô-las ao PBB, garante a privacidade.

7.3. Strong Timeliness

Na Seção 2.1, definimos *strong timeliness* como uma facilidade oferecida aos participantes para cancelar o protocolo de forma imediata e unilateral. As Figuras 15 e 16 ilustram como Alice e Bob utilizam essa propriedade no protocolo de sincronização.

A Figura 15 mostra os quatro caminhos que podem se desenvolver quando Alice posta primeiro (veja também a Figura 12). Outro conjunto de quatro caminhos semelhantes se desenvolve quando Bob posta primeiro; não discutiremos esses caminhos. O símbolo "*" significa qualquer token postado por Alice ou Bob, ou nenhum token. Esses tokens são irrelevantes porque não têm efeito na obtenção do resultado final: ou `Exchange Canceled` ou `Exchange successful`.

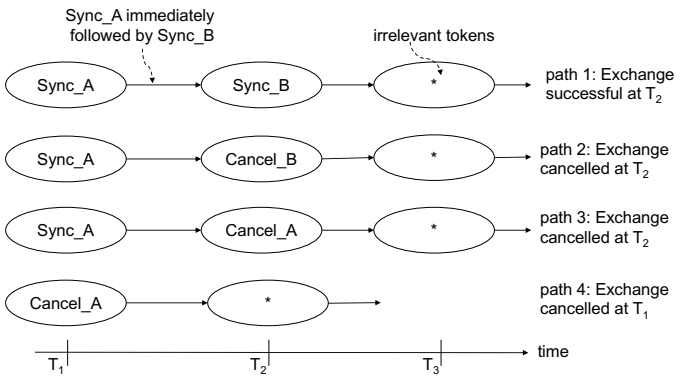

Figura 15. Post dos tokens de cancelamento para garantir timeliness.

A Figura 15 mostra que Alice pode usar o token `Cancel_A` para cancelar sob certas restrições que se materializam quando ela muda de ideia após postar `Sync_A`. Bob pode fazer o mesmo usando `Cancel_B`. Vamos examinar alguns desdobramentos. O *path* 4 ilustra que, se Alice não tiver postado nenhum token, ela pode postar `Cancel_A` para cancelar categoricamente; tokens postados em T_2 ou posteriormente, ou nenhum token adicional postado, não terão efeito sobre o resultado. O *path* 3 mostra um cancelamento sob restrição. Alice primeiro posta `Sync_A` para concordar com a troca; no entanto, ela muda de ideia e posta `Cancel_A`; O cancelamento de Alice somente produzirá efeito caso Bob ainda não tenha postado o seu token `Sync_B`. Se Bob postar `Sync_B` antes de Alice postar `Cancel_A`, `Cancel_A` torna-se irrelevante, como no *path* 1. O *path* 2 mostra o efeito imediato do cancelamento de Bob; Alice posta `Sync_A` cedo com a intenção de continuar a troca, mas `Cancel_B` anula seus planos.

A Figura 16–a mostra as linhas do tempo do *path* 2 da Figura 15. O intervalo antes de postar `Sync_A` em T_1 é seguro para Alice, ou seja, ela não corre risco de perder seu item. T_1–T_6 é arriscado para Alice: ela não pode cancelar o protocolo categoricamente nem abandoná-lo; precisa esperar até T_6. Ela está segura além de T_6, após perceber que Bob cancelou. Na verdade, Alice está segura a partir de T_4, mas ainda não sabe disso. T_2–T_4 é um intervalo de incerteza para Alice, cujo desenvolvimento depende de Bob. Bob permanece seguro o tempo todo: ele aproveita a *strong timeliness* em T_3 para cancelar e abandonar o protocolo sem se preocupar em recuperar os tokens.

A Figura 16–b mostra o *path* 3 em que Alice está segura antes de postar `Sync_A` em T_1; ela entra em um intervalo arriscado em T_1; executa a operação de *retrieve* em T_4, mas recupera apenas seu próprio token `Sync_A` porque Bob não postou nada. Alice repete (mostrado como "...") a operação de *retrieve* várias vezes com o mesmo resultado frustrante. Em T_5, ela perde a paciência e recorre à *strong timeliness* para se libertar da situação. Ela sai do intervalo arriscado em T_8, quando recupera `Sync_A`, `Cancel_A` e descobre que cancelou o protocolo. O intervalo de incerteza de Alice não é mostrado explicitamente, mas cobre T_1–T_8. Bob também aproveita a *strong timeliness*. Ele não executa nenhuma ação adicional e permanece permanentemente em estado seguro. O caso (a) ilustra como a nossa implementação simplificada impacta parcialmente a propriedade de *strong timeliness*: ela não permite a evolução para o cenário (b), pois não inclui a lógica necessária para que Alice execute a operação *retrieve* (T_4) de forma reiterada. A restrição pode ser removida, mas com custo de complexidade.

A Figura 17 apresenta evidências experimentais da propriedade de *strong timeliness* no protocolo proposto. No cenário ilustrado, Alice opta por cancelar a troca imediatamente, enviando seu token `Cancel_A`. Em seguida, ao realizar a operação *retrieve* no PBB, o seu *attestable* recebe o log contendo o token `Cancel_A`, comprovando que a solicitação de cancelamento foi registrada no PBB.

Por sua vez, Bob ainda não havia enviado o seu token, mas executa a operação *retrieve* antes disso. O resultado mostra que ele também obtém o mesmo log contendo o token `Cancel_A`, ou seja, o registro do cancelamento efetuado por Alice. Esse fato leva o *attestable* de Bob a finalizar no estado de "*Protocol Canceled*" também. Este resultado confirma que o protocolo implementado preserva a propriedade *strong timeliness*. Cada

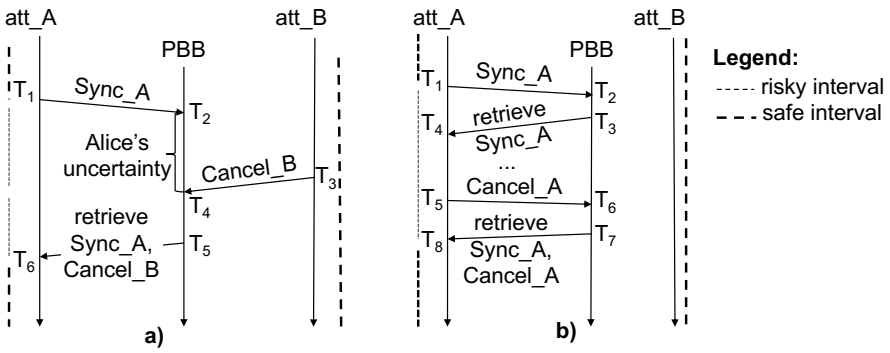

Figura 16. Timeliness com intervalos seguros, incertos e arriscados.

```
Retrieve operation was requested on the PBB.
...
...
...
...
The PBB sent the log to the Requester.
...
...
...
...
Alice's attestable received the log with the tokens: ["Cancel_A"] from the PBB.
Bob's attestable received the log with the tokens: ["Cancel_A"] from the PBB.
2025-10-22 20:58:55,388 INFO sqlalchemy.engine.Engine ROLLBACK
127.0.0.1 - - [22/Oct/2025 20:58:55] "GET /api/exchange/45528daa-40c5-4165-aaa0-d2edf94b3d2e/acceptance HTTP/1.1" 200 -
```

Figura 17. Alice Usa a propriedade Timeliness.

participante pode enviar seu *token* de cancelamento antes do protocolo atingir um dos dois estados finais, independentemente do comportamento da outra parte. Dessa forma, o sistema evita bloqueios indefinidos.

8. Trabalhos Relacionados

Nosso trabalho se inspira diretamente na descrição do *Fair Exchange Without Disputes* (FEWD) [Molina-Jimenez et al. 2024] e tem como objetivo fornecer a primeira implementação que demonstra que o FEWD é implementável com tecnologias atuais sem comprometer as propriedades que se esperam dele. Outra fonte de inspiração foi o trabalho de Avoine (veja, por exemplo, [Avoine and Vaudenay 2004]), no qual se sugere o uso de TEEs para depositar e bloquear documentos em troca. Nosso trabalho valoriza essa ideia e a avança com uma implementação prática. Além disso, no trabalho original, presumivelmente para evitar o uso de um TTP com estado, eles utilizam um protocolo probabilístico (KIT) para sincronização; como resultado, alcançam *weak fairness*, aceitável em algumas aplicações, mas não em outras. Este trabalho cobre as aplicações negligenciadas com a inclusão de uma parte sem estado (um PBB) para sincronização e fornecimento de *strong fairness*. Adicionalmente, nossa discussão sobre as propriedades dos protocolos de troca justa (*strong fairness*, *strong timeliness*) e sobre os itens em troca (idempotentes versus únicos) se baseia no trabalho de Asokan [Asokan et al. 2000], nosso PBB é uma versão simplificada de seu TTP *off-line*.

A partir dos resultados de Pagnia [Pagnia and Darmstadt 1999], aprendemos que, sem a participação de um TTP, um protocolo de troca não pode garantir *strong fairness*. Conclui-se que as propriedades de um protocolo de troca dependem de como o TTP, se

incluído, é utilizado. Vale ressaltar que, embora Pagnia não mencione explicitamente, ele se refere a um TTP usado para a execução da operação de sincronização (a operação crucial para alcançar consenso na troca); nós o chamamos de *synchronising* TTP. Implicitamente, ele também se refere a um TTP que, independentemente, é capaz de coletar informações sobre o estado global da troca; isso impede o uso de pares de TTPs remotamente separados em alguns protocolos. Usaremos a Tabela 1 para ilustrar esse ponto e resumir as propriedades dos protocolos que motivaram nosso trabalho.

Tabela 1. TTPs used in fair exchange protocols and protocols' properties.

Feature	[Brickell 1988]	[Pinkas 2003]	[Avoine 2004]	[Asokan 2002]	[Zhang 2024]	**FEWD**
Synchronizing TTP	–	–	–	✓	✓	✓
Split TTP	–	–	✓	–	–	✓
Strong Fairness	–	–	–	✓	✓	✓
Strong Timeliness	–	–	–	–	–	✓
Privacy	✓	✓	✓	–	–	✓

Os protocolos que não utilizam *synchronising TTPs* foram colocados no lado esquerdo da linha dupla. A quarta linha da tabela mostra que nenhum desses protocolos consegue garantir *strong fairness*. Nesses protocolos, a operação de sincronização é executada por *gradual release*. A inclusão do *gradual release* nesses protocolos compromete a propriedade *strong timeliness*: as partes não podem abandonar a execução do protocolo de *gradual release* a qualquer momento; uma vez iniciado, uma parte precisa executar o *gradual release* até a conclusão. Isso ocorre porque o *gradual release* é um protocolo *peer-to-peer* com estado, que progride gradualmente até os estados finais. A ausência de um *synchronising TTP* permite que esses protocolos garantam privacidade (veja a última linha da tabela) de forma mais simples e natural.

Os protocolos no lado direito da linha dupla utilizam um *synchronising TTP* e, portanto, são capazes de garantir *strong fairness*. A última linha da tabela mostra que nem Asokan 2002 nem Zhang 2024 podem garantir privacidade, devido ao uso de TTPs que executam, além da sincronização, operações que expõem informações sensíveis. Já o FEWD adota um TTP dividido em três componentes, incluindo o PBB, responsável exclusivamente pela operação de sincronização. Essa arquitetura favorece a garantia de privacidade e simplifica a implementação da *strong timeliness*; para ilustrar esse aspecto, pode-se comparar a operação de cancelamento apresentada na Figura 16b com aquela descrita em [Asokan et al. 2000].

9. Conclusões

A implementação do FEWD é desafiadora e complexa, pois o protocolo envolve múltiplas partes. A experiência inicial com a implementação do protocolo demonstrou a necessidade de modularizá-lo em partes menores e independentes. Para lidar com esta complexidade, seguimos o princípio dividir para conquistar [Horowitz and Zorat 1983]. Essa abordagem permitiu concentrar os esforços na operação de sincronização.

O protocolo completo pode ser separado em duas partes: uma dedicada ao protocolo de sincronização e outra voltada à manipulação de documentos. Neste artigo,

foi implementada exclusivamente a parte referente ao protocolo de sincronização, responsável pela operação *synchronise*. As demais operações, *deposit*, *verify* e *release/restore*, não foram implementadas nesta etapa e permanecem como trabalhos futuros. O FEWD foi projetado para prevenir disputas de não cumprimento e assegurar propriedades como *strong fairness*, *strong timeliness* e privacidade. Ao contrário da versão anterior [Quixabeira et al. 2025], esta versão apresenta uma implementação mais completa do protocolo de sincronização e uma análise formal de suas propriedades.

A primeira implementação, publicada no evento SBSeg 2025, teve caráter exploratório e seguiu uma abordagem direta, com levantamento mínimo de requisitos funcionais e não funcionais. Nesta segunda versão, o protocolo de sincronização foi analisado formalmente. Primeiramente, ele foi modelado como uma FSM e, em seguida, traduzido para a linguagem *Promela*, permitindo a verificação automática das propriedades desejadas, formalizadas por meio de fórmulas de LTL, utilizando o verificador *SPIN*. Essa abordagem baseada em *model checking* permitiu refinar o protocolo, assegurando maior precisão e rigor em relação à versão preliminar. Em seguida, o protocolo de sincronização foi implementado em Python, e os resultados obtidos confirmaram que as propriedades *strong fairness*, *strong timeliness* e privacidade também foram preservadas. A modelagem em *Promela* e a validação formal com o SPIN foram executadas em um laptop de tecnologia atual (Mac com CPU quad-core e memória padrão), sendo concluídas em média em 1 ou 2 segundos, sem consumo excessivo de memória ou saturação de recursos. Esses resultados evidenciam que o protocolo pode ser formalmente verificado com infraestrutura computacional acessível, reforçando sua aplicabilidade prática.

Além disso, a implementação foi realizada exclusivamente com tecnologias *open source*, utilizando Python e bibliotecas padrão, resultando em uma versão funcional enxuta, com código de complexidade moderada e viável de manutenção. Os experimentos conduzidos confirmaram que as propriedades de *strong fairness*, *strong timeliness* e preservação da privacidade foram mantidas também na implementação prática.

Em conjunto, os resultados demonstram que o protocolo não é apenas formalmente verificável, mas também plenamente implementável com tecnologias amplamente gratuitas e de código aberto. Essa característica está alinhada ao movimento de ciência aberta e à crescente ênfase na replicabilidade de experimentos, tendência de especial relevância para a área de Sistemas de Informação, pois favorece transparência, reprodutibilidade científica e adoção prática em diferentes contextos organizacionais.

Os resultados da verificação formal indicam que o protocolo FEWD satisfaz todas as propriedades especificadas em LTL, incluindo terminação, ausência de *deadlock* e preservação de justiça fraca. Em particular, a análise das FSMs confirmou que, independentemente da ordem de envio dos tokens `Sync` e `Cancel`, a execução sempre converge para um dos estados finais válidos, evitando situações de bloqueio indefinido. Esses achados demonstram que a utilização de um PBB stateless é suficiente para coordenar a sincronização sem comprometer a correção do protocolo, diferentemente de abordagens baseadas em TTPs stateful. Assim, os resultados obtidos reforçam que o FEWD se alinha às garantias teóricas de *fair exchange* discutidas na literatura, ao mesmo tempo em que reduz dependências de confiança e complexidade de implementação, pontos essenciais

para as organizações.

Quanto à aplicabilidade dos resultados aqui apresentados, podemos considerar vários contextos de aplicação. Um deles seria o das plataformas de *e-commerce*, nas quais a troca justa de informações é essencial para garantir que pagamentos, confirmações de entrega e acesso a bens digitais ocorram de forma sincronizada, sem que nenhuma das partes obtenha os dois itens. O protocolo proposto pode ser utilizado para coordenar essas interações, preservando a justiça e a privacidade dos participantes.

Outra possibilidade seria a aplicação em serviços de assinatura digital, nos quais o protocolo permite assegurar que documentos eletrônicos sejam assinados apenas quando todas as partes envolvidas concordam e as condições de troca são atendidas, evitando disputas. Já em ambientes colaborativos de troca de dados, como sistemas de *federated learning* ou parcerias entre organizações, o protocolo contribui para proteger dados sensíveis e de propriedade intelectual, garantindo que cada participante receba os benefícios acordados apenas quando os outros também cumprirem suas obrigações.

Esses cenários demonstram a aplicabilidade do protocolo em diversos domínios de sistemas de informação distribuídos, onde confiança, privacidade e justiça são essenciais. Também revelam seu potencial para reduzir a dependência de terceiros centralizados, promovendo maior autonomia e segurança nas transações digitais.

Disponibilidade dos Artefatos

A especificação em *Promela/SPIN* e o código-fonte que implementa o protocolo de sincronização está disponível em: `https://github.com/gca-research-group/fair-exchange`

10. Agradecimentos

Pesquisa parcialmente financiada pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), sob os projetos 311011/2022-5, 309425/2023-9 e 402915/2023-2.

Referências

[Asokan et al. 1997] Asokan, N., Schunter, M., and Waidner, M. (1997). Optimistic protocols for fair exchange. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 7–17. `https://dl.acm.org/doi/pdf/10.1145/266420.266426`.

[Asokan et al. 2000] Asokan, N., Shoup, V., and Waidner, M. (2000). Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in communications*, 18(4):593–610. `https://doi.org/10.1109/49.839935`.

[Avoine and Vaudenay 2004] Avoine, G. and Vaudenay, S. (2004). Fair exchange with guardian angels. In *International Workshop on Information Security Applications*, pages 188–202. Springer. `https://doi.org/10.1007/978-3-540-24591-9_15`.

[Brickell et al. 1988] Brickell, E. F., Chaum, D., Damgård, I. B., and van de Graaf, J. (1988). Gradual and verifiable release of a secret. In *Proceedings Advances in Cryptology*, pages 156–166. https://doi.org/10.1007/3-540-48184-2_11.

[Colletti 2017] Colletti, M. (2017). *Digital poetics: An open theory of design-research in architecture*. Routledge. https://doi.org/10.4324/9781315257761.

[Costan and Devadas 2016] Costan, V. and Devadas, S. (2016). Intel sgx explained. Cryptology ePrint Archive, Paper 2016/086. https://eprint.iacr.org/2016/086.

[Dierks and Rescorla 2008] Dierks, T. and Rescorla, E. (2008). Rfc 5246: The transport layer security (tls) protocol version 1.2. https://dl.acm.org/doi/pdf/10.17487/RFC5246.

[Fischer et al. 1985] Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382. https://doi.org/10.1145/3149.214121.

[Grisenthwaite et al. 2023] Grisenthwaite, R., Barnes, G., Watson, R. N. M., Moore, S. W., Sewell, P., and Woodruff, J. (2023). The arm morello evaluation platform—validating cheri-based security in a high-performance system. *IEEE Micro*, 43(3):50–57. https://doi.org/10.1109/MM.2023.3264676.

[Horowitz and Zorat 1983] Horowitz and Zorat (1983). Divide-and-conquer for parallel processing. *IEEE Transactions on Computers*, 100(6):582–585. https://doi.org/10.1109/TC.1983.1676280.

[Huang et al. 2014] Huang, Q., Wong, D. S., and Susilo, W. (2014). P 2 ofe: Privacy-preserving optimistic fair exchange of digital signatures. In *Topics in Cryptology–CT-RSA 2014: The Cryptographer's Track at the RSA Conference 2014, San Francisco, CA, USA, February 25-28, 2014. Proceedings*, pages 367–384. Springer. https://doi.org/10.1007/978-3-319-04852-9_19.

[Jarke et al. 2019] Jarke, M., Otto, B., and Ram, S. (2019). Data sovereignty and data space ecosystems. *Business & Information Systems Engineering*, 61(5):549–550. https://doi.org/10.1007/s12599-019-00614-2.

[Kaplan et al. 2016] Kaplan, D., Powell, J., and Woller, T. (2016). Amd memory encryption. *White paper*, 13:12. http://docs.amd.com/v/u/en-US/memory-encryption-white-paper.

[Lutsch et al. 2025] Lutsch, A., Franck, C., El-Hindi, M., István, Z., and Binnig, C. (2025). An analysis of aws nitro enclaves for database workloads. *Journal of the ACM*, pages 1–8. https://doi.org/10.1145/3736227.3736234.

[Markowitch et al. 2003] Markowitch, O., Gollmann, D., and Kremer, S. (2003). On fairness in exchange protocols. In *International Conference on Information Security and Cryptology – ICISC*, pages 451–465. https://doi.org/10.1007/3-540-36552-4_31.

[Markus and Silver 2008] Markus, M. L. and Silver, M. S. (2008). A foundation for the study of it effects: A new look at desanctis and poole's concepts of structu-

ral features and spirit. *Journal of the Association for Information systems*, 9(10):5. 10.17705/1jais.00176.

[Molina-Jimenez et al. 2024] Molina-Jimenez, C., Toliver, D., Nakib, H. D., and Crowcroft, J. (2024). *Fair Exchange: Theory and Practice of Digital Belongings*. World Scientific. https://doi.org/10.1142/9781800615175_0001.

[Pagnia and Darmstadt 1999] Pagnia, H. and Darmstadt, F. C. G. (1999). On the impossibility of fair exchange without a trusted third party. darmstadt university of technology. Technical report, Darmstadt University of Technology - Department of Computer Science. https://api.semanticscholar.org/CorpusID:11671049.

[Pinkas 2003] Pinkas, B. (2003). Fair secure two-party computation. In *Proceedings International Conference on the Theory and Applications of Cryptographic Techniques*, pages 87–105. https://doi.org/10.1007/3-540-39200-9_6.

[Pinto and Santos 2019] Pinto, S. and Santos, N. (2019). Demystifying arm trustzone: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 51(6):1–36. https://doi.org/10.1145/3291047.

[Quixabeira et al. 2025] Quixabeira, D., Teles-Borges, M., Roos-Frantz, F., Frantz, R. Z., Sawicki, S., Molina-Jimenez, C., and Crowcroft, J. (2025). Implementation and analysis of a synchronisation protocol for fair exchange with strong fairness and privacy. In *Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg)*, pages 498–514. SBC. https://doi.org/10.5753/sbseg.2025.11390.

[Zhang et al. 2024] Zhang, L., Kan, H., Qiu, F., and Hao, F. (2024). A publicly verifiable optimistic fair exchange protocol using decentralized cp-abe. *The Computer Journal*, 67(3):1017–1029. https://doi.org/10.1093/comjnl/bxad039.