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], Carlos Molina-Jimenez[2], Jon Crowcroft[2]

[1]Unijuí University – Ijuí, RS – Brazil

[2]University of Cambridge, Cambridge, UK
{dhileane.rodrigues, mailson.borges}@sou.unijui.edu.br
{frfrantz,sawicki,rzfrantz}@unijui.com.br
{carlos.molina,jon.crowcroft}@cl.cam.ac.uk

Abstract. *Strong fairness ensures that in fair exchange protocols, items are either exchanged or remain with their original owners. Privacy, another key property, prevents the leakage of sensitive information. Achieving both simultaneously is challenging. Existing protocols that ensure strong fairness, such as those in online payments, fail to guarantee privacy due to reliance on monolithic trusted third parties. These entities perform critical actions that expose data. We propose replacing them with a split trusted third party model, composed of two trusted execution environments (one per participant) and a public bulletin board used solely for synchronisation. Our analysis and implementation show that this approach preserves strong fairness, strong timeliness and privacy.*

1. Introduction

Fair Exchange Protocols (FEPs) enable two participants—say, Alice, in possession of an item, and Bob, in possession of another item—to swap them. The nature of the items varies and includes digital items (e.g. a photo or online accounts) that can be sent and received online, physical items (e.g. a bottle of wine or a pizza) [Molina-Jimenez et al. 2024], or a combination of both. For simplicity, in this paper, we will consider that the items are digital and can be regarded as digital strings, which we call digital items. This assumption preserves the main ideas that we wish to explain. Additionally, although we do not discuss physical items further, it is worth noting that in our protocol, they can be represented (modelled) by digital items. For example, PIN numbers that open physical containers can be exchanged with our protocol.

Several fair exchange protocols have been suggested that guarantee different properties, including fairness—the most fundamental property [Asokan et al. 1997], [Brickell et al. 1988], [Asokan et al. 2002], [Pinkas 2003], [Avoine and Vaudenay 2004], [Zhang et al. 2024]. Intuitively, when fairness is guaranteed, disputes never emerge because the protocol always produces fair outcomes. As elaborated in Section 2, protocols that guarantee strong fairness are called strong fair exchange protocols (SFEPs). A serious drawback of existing SFEPs is that they do not guarantee privacy. They leak sensitive information (for example, the nature of the item) to third parties, particularly to the trusted third parties (TTPs) that these protocols mandatorily need to involve [Pagnia and Gartner 1999]. Fair exchange is hard to solve because it is a fundamental distributed system problem that manifests recurrently in several online scenarios, such as payment versus delivery in online purchases, exchange of contract signatures, delivery

versus receipt in certified email, and bartering in cashless economies [Markowitch et al. 2003], [Asokan et al. 2002].

Representative examples of SFEPs that fail to protect privacy include those used on platforms such as Amazon, PayPal, and Alipay [Lou 2023] and [Winn and Wright 2000]. In some applications, the disclosure of sensitive information, such as shopping details, is merely a minor inconvenience that may be tolerated. However, some exchanges, such as the exchange of scientific data with business or intellectual value (for example, in federated learning), require privacy guarantees. We argue that the inability of existing SFEPs to provide privacy is due to the approach that they take in the implementation of the TTP: they use a monolithic TTPs that is application-aware, stateful, and responsible for manipulating sensitive information [Asokan et al. 1997], [Asokan et al. 2002], [Almutairi and Nigel 2019].

We argue that, with previous technologies, this approach was a justifiable compromise. However, the emergence of hardware enhanced for creating trusted execution environments (TEEs) and ubiquitous communication has motivated us to replace the monolithic TTP with a split TTP [Molina-Jimenez et al. 2024]. In this approach, the TTP is composed of three components: two TEEs deployed on Alice's and Bob's devices (one on each) and a Public Bulletin Board (PBB). With a split TTP-based solution, the execution of the operations involved in the fair exchange can be thoughtfully allocated to the most suitable component (TEEs or PBB) to achieve the desired property. For example, to guarantee strong fairness, the PBB is used for executing synchronisation; to guarantee privacy too, the PBB is stateless and application–agnostic; therefore, the PBB never has access to sensitive information. The latter is always kept within the confines of TEEs that remain under the control of their owners. This thoughtful allocation of operations contrasts with the monolithic TTP-based solution, where all operations are indiscriminately executed in the TTP, including those that handle sensitive information.

The main contribution of this paper is a demonstration of how to implement the synchronization module (a protocol in its own right) for fair exchange that guarantees strong fairness and privacy. We use Finite State Machines (FSMs) and timelines of message exchanges to analyse the properties of our protocol.

The remainder of this paper is structured as follows: Section 2 provides terminology and definitions. In Section 3, we discuss Fair Exchange Without Disputes (FEWD), the specification of the fair exchange protocol that has inspired our work. In Section 5, we describe our Python implementation of FEWD's operations. In Section 6, we demonstrate how our implementation of the synchronisation operation preserves strong fairness, privacy and other properties. Section 7 summarises research that has influenced our work. Section 8 discusses the main pending tasks that we have identified. Finally, in Section 9, we share our implementation experience and ideas stimulated by our results.

2. Background

Fair exchange protocols are executed between two parties, say Alice and Bob who are in possession of items `D_A` and `D_B`, respectively, to exchange their items. Fair exchange as a research topic gained broad attention in the early 80s thanks to Asokan's work [Asokan et al. 1997]. Since then, several protocols have been published that can be used to exchange items of different nature (e.g, digital, physical) and under different properties.

The most fundamental property of FEPs is fairness. From this perspective, they can be

divided into two classes:

- **Strong FEPs:** guarantee strong fairness, that is, upon completion of the protocol, either Alice is left in possession of `D_B` and Bob in possession of `D_A` or the items remain with their original owners. These protocols produce only fair outcomes and therefore prevent the occurrence of disputes.
- **Weak FEPs:** guarantee only weak fairness, that is, they might produce unfair outcomes where one of the parties is left with the two items and its counterpart gets none. Therefore, these protocols account for the potential occurrences of disputes. These are sorted out by dispute resolution mechanisms executed separately, that is, outside the normal execution of the protocol and are very likely to include humans acting as arbiters.

In 1999, Pagnia and Gartner demonstrated that fair exchange is at least as hard as consensus [Pagnia and Gartner 1999]; therefore, strong fairness cannot be achieved without the involvement of a TTP. The TTP is a central controller used for keeping the invariant that the FSMs of both participants reach the same final states. The salient feature of weak FEPs is that they can be implemented without TTPs (see Section 7).

2.1. Desirable Properties of Fair Exchange

In addition to strong fairness, there are other properties to evaluate a protocol. We will mention only the most relevant ones: privacy, timeliness, physical timeout independence [Asokan et al. 1997], [Markowitch et al. 2003], [Huang et al. 2014].

Privacy guarantees that only Alice and Bob have access to sensitive information. Timeliness ensures that a participant, say Alice, can unilaterally and immediately cancel the protocol, that is, without the need to wait for additional messages from Bob either directly or indirectly through a TTP. It is worth clarifying that to cancel, Alice might need to send or receive a message produced locally by the TTP. Physical timeout independence is observed when the protocol does not use physical clocks to timeout.

The challenge is to guarantee these properties without compromising strong fairness. For example, timeliness is trivial to implement without strong fairness—Alice can simply abandon the protocol at any time and lose her item. In this paper, we focus on strong fairness, privacy and timeliness. Additional information is in [Molina-Jimenez et al. 2024], [Almutairi and Nigel 2019], [Ray et al. 2005].

2.2. The Four Operations Included in Fair Exchange

A close examination of fair exchange will reveal that any fair exchange protocol can be separated into four basic operations: deposit, verify, synchronisation, and release/restore [Molina-Jimenez et al. 2024]. We will describe the operations from Alice's side; Bob's executions mirror Alice's. **Handshake** is executed by Alice and Bob to agree on the terms and conditions of the exchange, including items' descriptions. **Deposit** is executed by Alice to surrender her item. **Verify** is executed by Alice against Bob's item to check that the item is the one that she is expecting. **Synchronisation** is executed to determine whether the exchange will take place or cancelled. Finally, **Release/Restore** or nothing is executed: Release is executed to release Bob's item to Alice when the protocol completes in success. However, the execution of Release might be followed by the execution of Restore to return Alice's item (or compensation) to Alice, when the protocol completes in cancellation. In some protocols like FEWD, cancellation generates no operation; both items remain locked forever within TTPs.

In fact, all existing FEPs include these operations, not necessarily clearly identified and separated. We have observed that the approach taken to implement and execute these operations determines what properties (see Section 2.1) the protocol will be able to deliver. For example, in FEPs that use gradual release, synchronisation is executed between the two participants rather than in a TTP, therefore, gradual release protocols can guarantee only weak fairness. Similarly, in escrow-based protocols, deposit is executed in a stateful TTP; consequently, these protocols are unable to guarantee privacy.

2.3. TTP Architecture: Monolithic *versus* Split

A salient feature of existing strong FEPs is that they implement their TTPs following a monolithic architecture, as shown in Figure 1a; typically, the TTP is a server with storage and computational facilities. As shown in Figure 1a, the TTP is used for executing all the four basic operations (highlighted in bold to indicate where each operation is performed in both the Traditional Monolithic TTP and the Split TTP) discussed in Section 2.2. The appeal of such an architecture is its simplicity and familiarity; unfortunately, it compromises privacy.

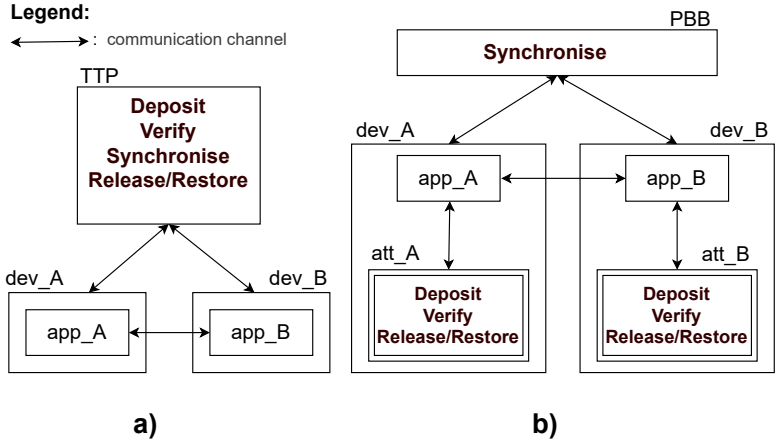

Figure 1. (a) Traditional monolithic TTP (b) Split TTP

Another alternative is to use a split architecture, shown in Figure 1b. The TTP is composed of three components: two attestables (one for each participant) and a PBB. The attestables are TEEs that comply with the attestable model discussed in Section 2.4. The attestables are shown in double lines to emphasise their security shield. They can be hardware–embedded in the device of each participant or somewhere else.

The PBB can be implemented by any public server (e.g. a web server, social network platform, etc.). Its functionality is similar to the blackboard architecture's [Long et al. 2017]. In our implementation, the main requirement on the PBB is that it is publicly available, offers API that accepts `post token` operations, has an arbitrarily large log for storing tokens permanently in some order, and accepts the `retrieve tokens` operations to deliver the whole log. A token is a string of characters, for example, "hi". The advantage of this approach is modularity. As shown in Figure1b, the execution of the four basic operations (shown in bold) can be thoughtfully allocated to the three components to achieve different properties. This is the approach taken by FEWD and we will elaborate in Section 3.

2.4. The Attestable Model and Current Implementation Technologies

An attestable is a model for executing code that manipulates sensitive data. It is expected to offer an API and observe three properties: Firstly, it is an execution environment within a black box that conceals data and computation from outside. Secondly, once a program is launched into execution, it will follow its logic faithfully, which might include absolute execution independence. Thirdly, it can attest to those first two points. The model can be implemented using different hardware and software technologies. For example, we can use ARM TrustZone [Pinto and Santos 2019], Intel SGX [Costan and Devadas 2016], AMD SEV [Kaplan et al. 2016], Amazon Nitro enclaves [AmazonAWS 2025] and compartments created in Morello Boards [ARM limited 2022]. These technologies are becoming common in conventional mobile devices, servers, and cloud platforms.

3. FEWD and the Main Four Operations of Fair Exchange

FEWD is a strong FEP that uses a split TTP. Its architecture is shown in Figure 2. Details on concepts that can aid in implementing this type of protocol can be found in [Molina-Jimenez et al. 2024]. Here, we present only a summary focusing on the concepts needed to understand the implementation discussed in Section 5.2.

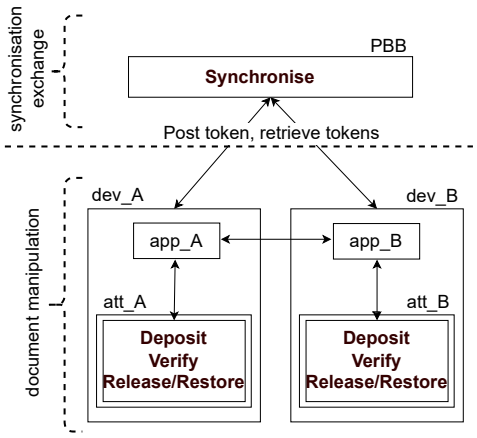

Figure 2. FEWD and its split TTP

Alice and Bob are in possession of personal devices `dev_A` and `dev_B`, respectively. Each device has access to an attestable (`att_A` and `att_B` respectively). In Figure 2, the attestables are shown in double lines to emphasise their security shield. They are shown embedded in the devices, but they can be somewhere else, for example, in the cloud. Alice and Bob initiate and drive the execution of the protocol from their respective applications (`app_A` and `app_B`). The attestables are assumed to have cryptographic facilities for building secure channels to communicate with the PBB and with each other.

We will use a simple scenario to explain the main ideas where Alice and Bob conduct the handshake operation offline. We use the following notations:

`D_A`: Alice's item, `D_B`: Bob's item, `[D_A]`: Alice's encrypted item, `[D_B]`: Bob's encrypted item, `att_A`: Alice's attestable, `att_B`: Bob's attestable, `app_A`: Alice's application, `app_B`: Bob's application, `Sync_A` and `Cancel_A`: Alice's tokens, `Sync_B` and `Cancel_B`: Bob's tokens. The main four operations are described as follows:

1. **deposit:** Alice and Bob deposit encrypted versions of their items in each other's attestables. As a result, after decryption `D_A` is locked in `att_B` and `D_B` is locked in `att_A`.
2. **verify:** `att_A` verifies the properties of `D_B`. On Bob's side, `att_B` verifies the properties of `D_A`. A party unsatisfied with the verification result either silently abandons or cancels the protocol.
3. **synchronise:** `att_A` posts a `Sync_A` token to the PBB to express her interest in continuing the protocol. Similarly and independently, `att_B` posts `Sync_B`. To learn the outcome, `att_A` and `att_B` independently, retrieves the log of tokens from the PBB.
4. **release:** If `att_A` retrieves `Sync_A` and `Sync_B` from the log, it releases `D_B` to `app_A`. Consequently, on Bob's side `att_B` will also retrieve `Sync_A` and `Sync_B` and release `D_A` to `app_B`. Observe that in FEWD the execution of restore is never needed. No operation is executed when the protocol is cancelled.

In summary, items are released through the execution of the `release` operation to their new owners applications (`app_A` and `app_B`) only when Alice posts `Sync_A` and Bob posts `Sync_B`. Items not released remain locked forever within the attestables. As explained in [Molina-Jimenez et al. 2024], `restore` operations are needed only in the exchange of some classes of items; we do not cover this topic in this paper. Items remain locked when Alice, Bob or both cancel the exchange. Figure 5 shows all possible developments of the protocol, including cancellations.

As shown in Figure 2, from the point of view of its functionality, FEWD is divided into two main parts: document manipulation and synchronisation. In the document manipulation part, operations such as deposit, verify, release/restore of items are performed, coordinated by the devices and their trusted environments. This part maintains the protocol state and handles sensitive information. In the synchronisation part, tokens are generated and exchanged to guide the outcome of the transaction between the involved parties. The latter is used by Alice and Bob only to produce a binary outcome—1 or 0—that, in FEWD, are interpreted as release or lock both items, respectively. The subtlety is that this outcome can be produced without any knowledge of the other parts of the protocol. As demonstrated in Section 6, the separate synchronisation operation enables privacy and other properties.

4. Example of Post of Sync and Cancel Tokens

Figure 3 illustrates two possible outcomes of the FEWD protocol: (a) `Exchange successful` and (b) `Exchange cancelled`. In (a), the protocol executes the following sequence: (1) - Alice's attestable (`att_A`) posts a `Sync_A` token to the PBB; (2) - Bob's attestable (`att_B`) posts a `Sync_B` token to the PBB; (3) Alice's attestable retrieves both tokens `Sync_A` and `Sync_B` from the PBB; (4) Bob's attestable also retrieves both tokens `Sync_A` and `Sync_B`. As a result, both parties confirm mutual commitment to proceed and the protocol reaches the `Exchange successful` state.

In (b), the protocol executes the following sequence: (1) - Alice's attestable initiates the exchange by posting `Sync_A` to the PBB on the hope to complete the exchange in success; (2) - In contrast, Bob's attestable decides to cancel the protocol and posts `Cancel_B`; (3) Alice's attestable retrieves both tokens `Sync_A` and `Cancel_B` from the PBB; (4) - Bob's attestable also retrieves both tokens `Sync_A` and `Cancel_B` from the PBB. As a result, the protocol reaches the `Exchange cancelled` state.

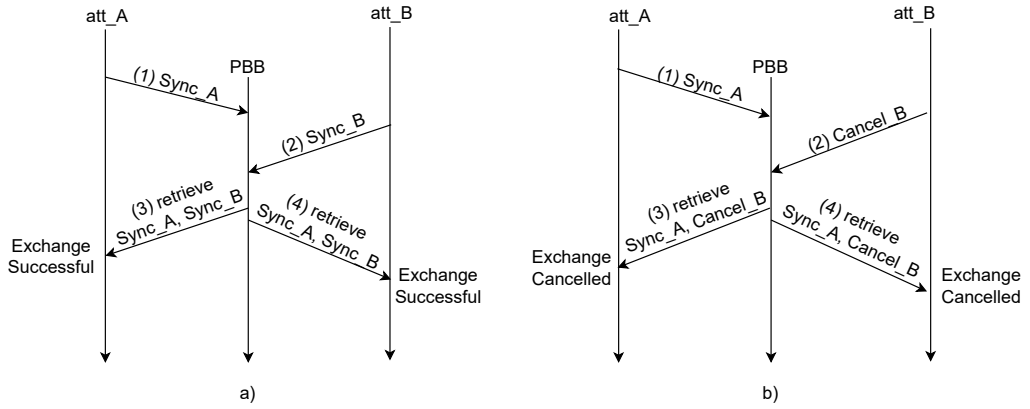

Figure 3. Outcomes of FEWD: (a) Exchange successful; (b) Exchange cancelled.

5. Implementation of FEWD

The specification of FEWD was published in 2024 [Molina-Jimenez et al. 2024], but no implementation was reported. A complete implementation is time demanding. Luckily, the protocol can be cleanly separated into two parts (see Section 3) that can be implemented separately to scrutinise their properties. Our article focuses on the synchronisation protocol that is executed with the help of tokens posted to and retrieved from a PBB.

5.1. Implementation of the Handshake and Initial Setup

To start the protocol, Alice and Bob execute a handshake operation to create a setup document that they agree to observe during the execution of the protocol. To this end, the attestables create their public keys and define the user identities. The setup document stipulates that Alice and Bob agree to the terms of the exchange. Although the setup document can include several parameters, our implementation includes only three:

- **PBB:** specifies the uniform resource locator (URL) and transfer layer security (TLS) [Dierks and Rescorla 2008] certificate of the PBB that Alice and Bob agree to use.
- **Device keys:** specifies the keys and certificates to be used for the attestation of Alice's and Bob's trusted components.
- **Document:** describes the items that Alice and Bob are exchanging. It contains three sub-parameters: **setup_values**, **module**, and **certificate**. The module contains four sub-parameters: **setup_values_types**, **doc_values_types**, **verify_function**, **gather_function**. Among them, the **gather_function** is responsible for generating the **doc_values**.

Let us examine the description of Alice's item. The description is symmetrically applied to Bob as well. For instance, `M_A`, refers to the module in Alice's item description, and `M_B` refers the module in Bob's description.

The setup_values, denoted by `P_A`, are the values provided by Alice. Their corresponding types form the list of setup_values_types, represented as `S_A`. The `doc_values_types`, denoted T_A, refer to the types expected by Bob. The gather_function collects the values of the original item from Alice. We use the notation G_A. The values collected are put in a list of doc_types, that is, $X_A = \{x_1, \ldots, x_M\}$. If the types of the values in X_A match the types listed

in doc_values_types, the gather function outputs X_A, otherwise it outputs an error, witch cancels the exchange. The verify_function is responsible for validating the data. It takes setup_values from the setup document and the doc_types list gathered by the gather function as input and produces a boolean: True if the values in setup_values match the values in doc_types. True and false represent, respectively, the acceptance and rejection of `D_A`.

Imagine that Alice and Bob are willing to exchange items using our protocol. Bob expects to receive a book. Algorithm 1 shows an example of a handshake and Initial Setup Description. This algorithm contains the list setup_values, setup_values_types and doc_values_types, P_A, S_A and T_B, respectively. It also shows the function verify and gather, V_A and G_A, respectively.

Algorithm 1: Handshake and Initial Setup Description

```
   // Setup_values
1  P_A ← ["png", 10, "Alice"]                          // P_A ← [P_1, P_2, P_3]
   // setup_values_types
2  S_A ← [format, sizeMB, author]                      // S_A ← [S_1, S_2, S_3]
   // doc_values_types
3  T_B ← [format, sizeMB, author]                      // T_A ← [T_1, T_2, T_3]
   // gatherer function
4  Function Gather_function (D_A):
5      x_1 ←D_A.format
6      x_2 ←D_A.sizeMB
7      x_3 ←D_A.author
8      return [x_1, x_2, x_3]
                                                       // X_A
   // the verify function
9  Function Verify_function:
10     P_1 : S_1,   P_2 : S_2,   P_3 : S_3,
11     X_1 : T_1,   X_2 : T_2,   X_3 : T_3,
12 if x_1 ≠ p_1 then
13     return false
14 if x_2 > p_2 then
15     return false
16 if x_3 ≠ p_3 then
17     return false
18 return true
```

5.2. Implementation of the Synchronisation Operation

We are currently using Python to implement the synchronisation part to examine how strong fairness and privacy are guaranteed. The code is available from `https://github.com/gca-research-group/fair-exchange-v1`. We have implemented the attestables as TCP clients, namely, `att_A.py` and `att_B.py`. The PBB, Alice's application and Bob's applications have been implemented as TCP servers, `PBB.py`, `app_A.py` and `app_B.py`, respectively.

Figure 4 shows the modules of our current implementation where `dev_A` and `dev_B` represent Alice's and Bob's devices, respectively. We have used TCP servers and clients that communicate with each other over secure channel ($secchan$) created with TLS, to realise the components shown in Figure 2. In this version, the whole code is executed by a single user (rather than two independent users with their own devices) using the menu shown in the fig-

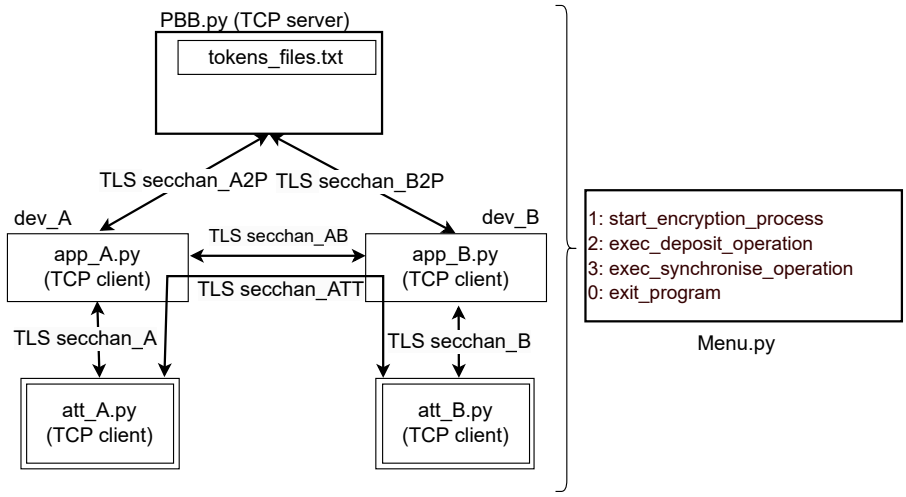

Figure 4. Python implementation of FEWD

ure. Option "1" causes `app_A` and `app_B` to encrypt, respectively, `D_A` and `D_B`. Option "2" triggers the execution of `deposit`: `[D_A]` and `[D_B]` are locked in`att_B` and `att_A`, respectively. Option "3" triggers the execution of `synchronise` and invites Alice and Bob to type and submit their tokens: `Sync_A` or `Cancel_A` and `Sync_B` or `Cancel_B`, respectively. The database used in our PBB stores an ID that identifies the exchange, along with the user's token.

To simplify the code, we have introduced the following slight diversions from the original specification described in [Molina-Jimenez et al. 2024]:

1. In the original specification, the post and retrieve operations use independent communication sessions. Our implementation uses a single communication session that is used for posting and retrieving.
2. In the original specification, the attestables can repeatedly post tokens `Sync` and `Cancel` and are responsible for scanning the retrieved log. In our implementation, a token can be posted only once.
   Unbounded token postings generates several deployment difficulties. It exposes the system to denial-of-service (DoS) attacks, where a malicious participant can flood the protocol with excessive token submissions (e.g. `Sync_A`, `Sync_A`, `Sync_A`, etc.). Additionally, redundant token processing increases computational and storage overhead that degrades performance. Semantic ambiguity is another concern —multiple instances of the same token type complicate validity checks. Moreover, repeated submissions can be exploited to delay the completion of the protocol.
3. In the original specification, all the participants are allowed to read the PBB's log. The PBB returns the entire log that includes all received tokens. However, our implementation constraints the PBB to return only the tokens related to the current execution instance. In other words, only the participants directly involved in a specific exchange have access to the tokens.

We are aware that these diversions result in different properties. Yet, as demonstrated in Section 6, our implementation retains strong fairness, privacy and timeliness.

5.3. Implementation of the Attestables

The attestables `att_A.py` and `att_A.py` are implemented as TCP client. Incidentally, `att_A`, interacts with `app_A`, `app_B` and PBB. In algorithm 2, Alice's attestable receives the item `D_A` to encrypt it. Additionally, Alice's attestable accepts a connection via a secure channel from `att_B` through `app_A`, sends the encrypted file [`D_A`] to `att_B`, receives an encrypted file [`D_B`] from `att_B`, and verifies [`D_B`]. Next, the `att_A` posts `Sync_A` or `Cancel_A` to `PBB`. After retrieving the tokens from the `PBB` through `app_A`, `att_A` executes either release to surrender `D_B` to `app_A` or lock `D_B` forever.

Algorithm 2: Implementation of the attestables

```
1  Function ProcessAttestables(D_A ∈ 𝒟, D_B ∈ 𝒟):
2      encrypted_D_A ← Encrypt(D_A)
3      att_B ← Send(encrypted_D_A)
4      att_A ← Receive(encrypted_D_B)
5      isValid ← Verify(encrypted_D_B)
6      if isValid then
7          PBB ← Send(Sync_A)
8      else
9          PBB ← Send(Cancel_A)
10     tokens ← RetrieveTokens(PBB)
```

5.4. Implementation of the PBB

Being a server, the PBB remains listening for connection requests placed by clients `app_A` and `app_B` to post or retrieve tokens originally generated by `att_A` and `att_B`. The PBB stores received tokens in a list. It returns the list in response to retrieve operations. Algorithm 3 is an pseudocode excerpt from `PBB.py`.

Algorithm 3: Implementation of the PBB

```
1  Function ProcessPBB(request ∈ {receive, retrieve}, s ∈ {att_A, att_B, app_A, app_B}):
2      if indication.receive == ok then
3          if s = att_A or s = att_B then
4              PBB ← Receive(tokens);
5          else
6              Error: invalid sender for token reception;
7      else if request = retrieve then
8          if s = app_A or s = app_B then
9              PBB ← RetrieveRequest(s);
10             s ← ReturnTokens(tokens);
11         else
12             Error: invalid requester for token retrieval;
13     else
14         Error: invalid request type;
```

5.5. Implementation of Alice's Application

Alice's `app_A.py` and Bob's `app_B.py` applications bridge the communications between their respective attestables and the PBB. For example, to post a token, `att_A.py` sends the token to `app_A` and `app_A` forwards it to the PBB. In a direct interaction, Alice's application sends the item to be encrypted by its attestable and receives either the decrypted item or a notification indicating that the exchange has been aborted. Algorithms 4, shows a pseudocode excerpt of the implementation of `app_A.py`.

Alice's application sends `D_A` to Alice's `att_A` for encryption. It forwards the items involved in the deposit operation between Alice's `att_A` and Bob's `att_B`. During the synchronisation process, Alice's application forwards tokens between `att_A` and PBB and receives either `D_B` or a cancel notification.

Algorithm 4: Behavior of Alice's Application (`app_A`)

```
1  Function Processapp_A (D_A ∈ 𝒟, token ∈ 𝒯) :
2      att_A ← Receive(D_A);
3      response ← ReceiveFrom(att_A);
4      if response = encrypted_D_A then
5          att_B ← Send(encrypted_D_A);
6      else
7          return AbortNotification;
8      token ← ReceiveFrom(att_A);
9      PBB ← Send(token);
10     result ← ReceiveFrom(PBB);
11     if result = valid then
12         return D_B;
13     else
14         return AbortNotification
```

6. Analysis of Properties

We have simplified the implementation of the PBB and traded generality for simplicity. We will discuss next what aspects of privacy are preserved or compromised in our current implementation.

6.1. Strong Fairness

The FSM of Figure 5 shows how strong fairness is preserved in our implementation. According [Molina-Jimenez et al. 2024], strong fairness can only be guaranteed in protocols where the synchronise operation is executed in an independent messages environment, i.e., a messaging service that cannot be controlled by the participants. In our implementation, this environment is provided by the PBB. The figure shows the FSM of the synchronise operation. It consists of five states: one initial, two intermediate and two final states. There are eight possible paths from the initial to the final states. Strong fairness is guaranteed because the FSM always progresses from the initial state to one of the two mutually exclusive final states; either `Exchange successful` or `Exchange cancelled`.

Figure 5 shows of the paths that progresses from the initial states to one of the final states, namely `Exchange successful`. It also shows of the paths that progresses from the initial state to the `Exchange cancelled` state, i.e., the final state where the exchange is cancelled.

6.2. Privacy

In SFEPs privacy in put at risk by the information that Alice and Bob reveal to the TTP. As explained in Section 2, the latter is needed at least for the execution of the synchronise operation [Pagnia and Gartner 1999]. Figure 5 shows how our implementation can execute the synchronise operation without revealing sensitive information to the PBB: Observe that the FSM uses only four application–agnostic strings as tokens: `Sync_A`, `Cancel_A`, `Sync_B` and `Cancel_B` to synchronise in either `Exchange cancelled` or

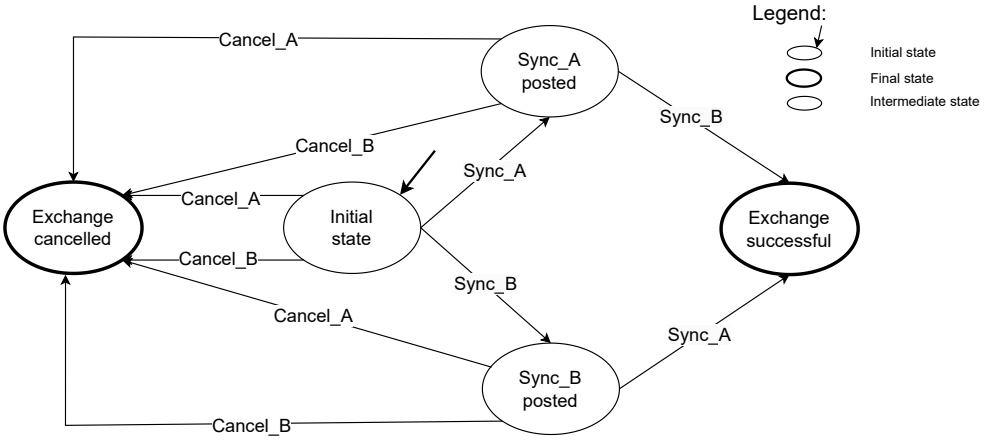

Figure 5. FSM of sync operation with its two unique and mutually exclusive final states.

`Exchange successful`. In practice, the strings might include some information meaningful to Alice and Bob (for example, keys and signatures), yet they do not need to include information about the items under exchange, the identities of parties involved or the development of the exchange. Therefore, they not need to be concealed from the PBB or from other parties that might have access to it. With this approach, the PBB is not only stateless but also application–agnostic. It is only a token repository that offers two services: it stores tokens and delivers tokens.

The description of the FEWD in [Molina-Jimenez et al. 2024], the PBB accepts a token from anybody and appends it to the tail of an arbitrarily large log. Upon request, it surrenders the whole log to anybody. This is certainly a general model of a PBB that can be used to synchronise with privacy guarantees. However, the implementation of such a PBB is demanding due to its generality; also the fact that the token requester always retrieves the whole log (perhaps with thousands of tokens) for scanning to determine if the two tokens that he needs are or are not yet in the log, questions the efficiency of the model. In our implementation, we have simplified the PBB log. We assume that Alice and Bob run only a single instance of the protocol. The PBB is responsible for receiving tokens and recording them in a database that is associated with a unique exchange ID between participants, ensuring that only tokens corresponding to that specific exchange are stored; therefore the tokens need to include the exchange's ID. This exchange's ID is generated during the initial handshake between the participants. The association of tokens to instances of the protocol simplifies the task of filtering tokens in the PBB and therefore simplifies the code that implements the PBB. It also frees the attestables from the task of token filtering.

We stress that, in our protocol, the deposit, verify and release/restore operations are executed within the attestables, while the synchronise operation is executed in the PBB. The use of attestables for executing sensitive document–related operations, without exposing them to the PBB, guarantees privacy.

6.3. Strong Timeliness

We defined timeliness in Section 2.1 as a facility offered to the participants to cancel the protocol immediately and unilaterally. We will use Figures. 6 and 7 to explain how timeliness can

be used by Alice and Bob in FEWD.

Figure 6 shows the four paths that can develop when Alice posts firstly (see also Fig. 5). Another set of four similar paths develops when Bob posts firstly, we do not discussion them. The symbol "*" means any token posted by Alice or Bob or no token posted at all. These tokens are irrelevant because they have no effect in reaching the final outcome: either `Exchange cancelled` or `Exchange successful`.

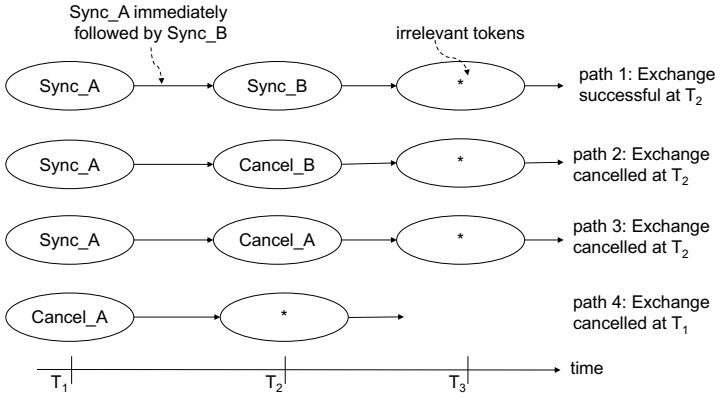

Figure 6. Post of cancel tokens to guarantee timeliness at synchronisation.

Figure 6 shows that Alice can use token `Cancel_A` to cancel, admittedly, under certain restrictions that materialise when she changes her mind after posting `Sync_A`. Bob can do the same using `Cancel_B`. Let us examine some developments. Path 4 illustrates that if Alice has not posted any token, she can post `Cancel_A` to cancel categorically; tokens posted at T_2 or later or no additional tokens posted will have no effect on the outcome. Path 3 shows a cancellation under a restriction. Alice firstly posts `Sync_A` to agree to the exchange; however, she changes her mind and posts `Cancel_A`; Alice's cancellation takes effect only because Bob has not posted `Sync_B` yet. If Bob posts `Sync_B` before Alice posts `Cancel_A`, `Cancel_A` becomes irrelevant as in path 1. Path 2 shows the immediate effect of Bob's cancellation; Alice posts `Sync_A` early with the intention to continue the exchange, but `Cancel_B` spoils her plans.

Figure 7–a shows the timelines of path 2 of Figure 6. The interval before posting `Sync_A` at T_1 is safe for Alice, i.e., she is not at risk of losing her item. T_1–T_6 is risky for Alice: she cannot cancel the protocol categorically or abandon it. She needs to wait until T_6. She is safe beyond T_6, after learning that Bob has cancelled. In fact, Alice is safe from T_4 onwards, but she does not know yet. T_2–T_4 is an interval of uncertainty for Alice, the development depends on Bob. Bob remains safe all the time: he takes advantage of timeliness at T_3 to cancel and abandon the protocol without bothering to retrieve the tokens.

Figure 7–b shows path 3 where Alice is safe before posting `Sync_A` at T_1; she enters a risky interval at T_1; she executes the retrieve operation at T_4 but she retrieves only her own `Sync_A` token because Bob has posted nothing. Alice repeats (shown as "...") retrieve several times with the same disappointing result. At T_5 she loses her patience and resorts to timeliness to free herself from the situation. She leaves the risky interval at T_8 when she retrieves `Sync_A`, `Cancel_A` and learns that she has cancelled the protocol. Alice's interval of uncertainty is not shown explicitly but covers T_1-T_8. Bob takes advantage of timeliness too. He does absolutely

nothing and remains always safe. b) is an example of how our simplified implementation partially impacts timeliness: it cannot develop b) because it does not include code for Alice to execute retrieve (T_4) repeatedly. The restriction can be lifted but at complexity cost.

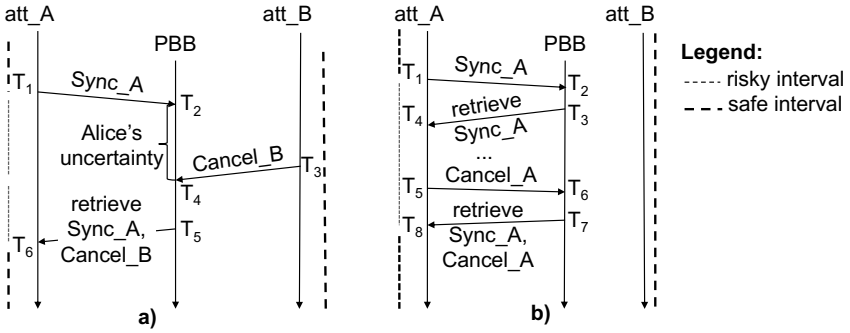

Figure 7. Timeliness with safe, uncertain and risky intervals.

7. Related Work

Table 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	✓	✓	✓	–	–	✓

Our work takes inspiration directly from the description of Fair Exchange Without Disputes (FEWD) [Molina-Jimenez et al. 2024] and aimed at providing the first implementation to demonstrate that FEWD is implementable with current technologies without compromising the properties that it is expected to deliver. Another source of inspiration was Avoine's work (see for example [Avoine and Vaudenay 2004]) where the use of trusted execution environments is suggested to deposit and lock documents under exchange. Our work appreciates the idea and progresses it with actual implementation. Also, in the original work, presumably to avoid the use of a stateful TTP, they use a probabilistic protocol (KIT) to synchronise, as a result, they achieve weak fairness, which is acceptable in some applications but not in others; we cover the neglected applications with the inclusion of the stateless party (a public bulletin board) to synchronise and provide strong fairness. In addition, our discussion on the properties of fair exchange protocols (strong fairness, timeliness) and of the items (idempotent versus unique) under exchange is grounded on Asokan's work [Asokan et al. 2002] — our PBB is a simplified version his off–line TTP.

From Pagnia's results [Pagnia and Gartner 1999] we have learnt that without the involvement of a TTP an exchange protocol cannot guarantee strong fairness. It follows that the properties of an exchange protocol depend on how the TTP, if included, is used. It is worth remarking that though Pagnia does not mention it explicitly, he refers to a TTP used for the execution of the synchronisation operation (the crucial operation to achieve consensus in the

exchange); we call it a synchronising TTP. Also, implicitly, he refers to a TTP, that independently, is able to collect information about the global state of the exchange; this precludes pairs of remotely separated TTPs used in some protocols. We will use Table 1 to show the point and to summarise the properties of the protocols that have motivated our work. In the table, we have placed the protocols that do not use synchronising TTPs on the left side of the double line. The fourth row of the table shows that none of these protocols can guarantee strong fairness. An examination of the cited works will reveal that in these protocols, the synchronisation operation is executed by gradual release —a protocol in its own right. The inclusion of gradual release in these protocols compromise timeliness: the parties cannot abandon the execution of the gradual release protocol at arbitrary times, once initiated, a party needs to run the gradual release to completion. The reason for this is that gradual release is a peer–to–peer stateful protocol that gradually progresses to end states. The preclusion of a synchronising TTP allows these protocols to guarantee privacy (see the last row of the table) in a simpler and natural manner.

In contrast, the protocols on the right side of the double line use a synchronising TTP and, therefore, are able to guarantee strong fairness. The last row of the table shows that neither Asokan 2002 nor Zhang 2024 can guarantee privacy, due to the use of TTPs that execute (in addition to synchronisation) operations that expose to them sensitive information.

In contrast, thanks to the use of a split TTP that includes a component used only for the execution of the synchronisation operation, FEWD (our protocol) is able to guarantee privacy. It also guarantees timelines in a simpler manner, to appreciate this point, it might help to contrast the simplicity of the cancellation operation shown in Fig. 7 against the cancellation operation of [Asokan et al. 2002].

8. Future Work

Several implementation and fundamental research tasks remain pending. Incidentally, the synchronise operation in FEWD deserves further analysis. In Figure 4, it is implemented with the help of the PBB, as a result, it guarantees strong fairness and privacy; yet we are aware that other synchronisation mechanisms can be used to release or lock the items forever. Accordingly, we are planning to replace the PBB–based synchronisation by other synchronisation protocols and compare their properties and performance. In particular, we will implement and use protocols that guarantee only weak fairness like gradual release like the Keep in Touch protocol [Avoine and Vaudenay 2004]. As explained in [Molina-Jimenez et al. 2024], fair exchange raises legal implications that we are planning to study: for example, regarding privacy, what are the legal implications of using protocols like FEWD for exchanging illegal items? Our future work includes also the formal validation of the protocol (e.g. by model–checking) and systematic testing of the current and new versions of the implementation using different case studies. We also intend to conduct a deeper performance metrics analysis and carry out scalability testing. Another avenue on our agenda is to use a blockchain with a smart contract (in place of the stateless PBB) to implement the execution of the synchronisation operation; preliminary observations suggest that a PBB with storage and computation facilities for building a global view (in the PBB) of the protocol will simplify the storage and analysis of the token log and the whole protocol. The inclusion of a blockchain is likely to introduce further engineering and cryptographic challenges.

9. Conclusions

The related work section reveals that research on fair exchange was intense in the 80s but recently quite. Why are we re–birthing the topic right now.? Two reasons have motivated our work. Firstly, we believe that online transactions where items are swapped directly (i.e. without money mediation) will increase with data proliferation; in open science for example, researchers frequently exchange data, in social networks, participants frequently swap personal photos, addresses and locations. Secondly, we believe that the availability of technologies for instantiating TEE simplifies the fair exchange and other similar problems; and can help in building simpler protocols with better properties. The Python implementation discussed here consists of about 1570 LOC. To evaluate the proposed scheme, we conducted tests with results demonstrated that the proposed protocol is capable of ensuring strong fairness, privacy and timeliness. Our current results have raised questions about the security of the TEE; we can ask if the items left locked forever in the attestable when the protocol is cancelled, will persist inaccessible forever or only temporary safe, that is, till Bob hacks his attestable and extracts Alice's item. There are other open questions. For example, to what extent the description of items used in verify give away the items? From our implementation experience, this question seems more difficult than synchronisation. We leave these questions for future work.

Availability of artifacts

The source code that implements the synchronisation protocol of FEWD is publicly available at: `https://github.com/gca-research-group/fair-exchange-v1`.

Acknowledgements

Research partially funded by the Co-ordination for the Brazilian Improvement of Higher Education Personnel (CAPES) and the Brazilian National Council for Scientific and Technological Development (CNPq) under project grants 311011/2022-5, 309425/2023-9, 402915/2023-2. EPSRC/EP/X015785/1 (G115169) funded Carlos Molina and Jon Crowcroft. Thanks to Dann, Hazem and Mansoor from the Centre for Re-decentralisation (CRDC) of Univ. of Cambridge.

References

Almutairi, O. and Nigel, T. (2019). Performance modelling of an anonymous and failure resilient fair-exchange e-commerce protocol. In *Proceedings International Conference on Performance Engineering*, pages 5–12.

AmazonAWS (2025). AWS Nitro system. https://aws.amazon.com/ec2/nitro/nitro-enclaves/.

ARM limited (2022). ARM morello program. https://www.arm.com/architecture/cpu/morello.

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.

Asokan, N., Shoup, V., and Waidner, M. (2002). Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in communications*, 18(4):593–610.

Avoine, G. and Vaudenay, S. (2004). Fair exchange with guardian angels. In *International Workshop on Information Security Applications*, pages 188–202. Springer.

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.

Costan, V. and Devadas, S. (2016). Intel SGX explained. *Cryptology ePrint Archive*.

Dierks, T. and Rescorla, E. (2008). RFC 5246: The transport layer security (TLS) protocol version 1.2.

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.

Kaplan, D., Powell, J., and Woller, T. (2016). AMD memory encryption. *White paper*, 13:12.

Long, E. S., Nguyen, B., Pitropov, M. A., and Torres, E. A. (2017). Blackboard architecture style. https://student.cs.uwaterloo.ca/ cs446/1171/Arch_Design_Activity/Blackboard.pdf.

Lou, J. (2023). *Sensitive Data Risks Analysis in Emerging Online Platforms*. University of Louisiana at Lafayette.

Markowitch, O., Gollmann, D., and Kremer, S. (2003). On fairness in exchange protocols. In *International Conference Information Security and Cryptology—ICISC*, pages 451–465.

Molina-Jimenez, C., Toliver, D., Nakib, H. D., and Crowcroft, J. (2024). *Fair Exchange: Theory and Practice of Digital Belongings*. World Scientific.

Pagnia, H. and Gartner, F. C. (1999). On the impossibility of fair exchange without a trusted third party. Technical report, Darmstadt University of Technology.

Pinkas, B. (2003). Fair secure two-party computation. In *Proceedings International Conference on the Theory and Applications of Cryptographic Techniques*, pages 87–105.

Pinto, S. and Santos, N. (2019). Demystifying arm trustzone: A comprehensive survey. *ACM computing surveys (CSUR)*, 51(6):1–36.

Ray, I., Ray, I., and Natarajan, N. (2005). An anonymous and failure resilient fair-exchange e-commerce protocol. *Decision Support Systems*, 39(3):267–292.

Winn, J. K. and Wright, B. (2000). *The law of electronic commerce*. Wolters Kluwer.

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.