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

Dhileane de Andrade Rodrigues[1], Fabricia Roos-Frantz[1], Rafael Z. Frantz[1]
Sandro Sawicki[1], Carlos Molina-Jimenez[2] and Jon Crowcroft[2]

[1]Unijuí University – Ijuí, RS – Brazil

dhileane.rodrigues@sou.unijui.edu.br

{frfrantz,sawicki,rzfrantz}@unijui.com.br

[2]University of Cambridge – Cambridge – United Kingdom

{carlos.molina,jon.crowcroft}@cam.ac.uk

Abstract. *Strong fairness is a highly desirable property in fair exchange protocols; it guarantees that either the items are exchanged or remain with their original owners. Privacy is another desirable property — it guarantees that the protocol does not leak sensitive information to third parties. It has proven to be hard to guarantee both. Existing fair exchange protocols with strong fairness including those used in current online payment versus delivery, are unable to guarantee privacy. The reason is that they use monolithic trusted third parties that indiscriminately execute crucial operations that leak information. In this paper we claim that privacy can be guaranteed by replacing the monolithic trusted third parties by a split trusted third parties composed of two trusted execution environments (one in the device of each participant) and a public bulletin board. The latter is used only for synchronising the exchange and without leaking sensitive information. We show results from an implementation that demonstrate that strong fairness, privacy and other properties can be guaranteed with the split trusted third parties approach.*

Keywords: *Fair Exchange Protocol, synchronisation Protocol, Distributed Systems.*

1. Introduction

Fair exchange protocols (FEP) 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), 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 documents. This simplification is rich enough to explain the ideas in this paper. Additionally, although we do not discuss physical items further, it is worth noting that in fair exchange protocols, they are represented by digital documents, such as cryptographic keys to unlock them from containers. Fair exchange 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. 1998, Molina-Jimenez et al. 2020].

Several fair exchange protocols have been suggested that guarantee different properties, including fairness—the most fundamental property. Intuitively, fairness guarantees that disputes never emerge because the protocol always produces fair outcomes. As elaborated in Section 3, protocols that guarantee strong fairness are called strong fair exchange protocols (SFEP). 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 TTP that these protocols mandatorily need to involve [Pagnia and Darmstadt 1999].

Representative examples of SFEP that fail to protect privacy include those used on platforms such as Amazon, PayPal, and Alipay. 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 ML), require privacy guarantees.

We argue that the inability of existing SFEPs to provide privacy arises from the approach they take in implementing the TTP: they use a monolithic TTP that is application-aware, stateful, and responsible for manipulating sensitive information [Asokan et al. 1997, Asokan et al. 1998, Asokan et al. 2000, Almutairi and Nigel 2019]. We believe that, with previous technologies, this approach was a justifiable compromise. However, the emergence of hardware-enhanced for creating trusted execution environments and ubiquitous communication has prompted 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 trusted execution environments 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 (trusted execution environments or PBB) to achieve the desired property. For example, to guarantee strong fairness, the PBB is used for executing synchronisation; to also guarantee privacy, the PBB is implemented stateless and application-agnostic; therefore, the PBB never has access to sensitive information. The latter is always kept within the confines of trusted execution environments 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 synchronisation operation in fair exchange to guarantee strong fairness and privacy. We contribute the Python code (available from GitHub) that we use to demonstrate our claims. We use execution outputs to support the FSMs and timelines that we employ in the analysis of the properties.

The remainder of this paper is structured as follows: Section 2 summarises research that has influenced our work. Section 3 provides terminology and definitions. In Section 4, we discuss FEWD, the specification of a fair exchange protocol that has inspired our work. In Section 5, we describe our Python implementation of the synchronisation operation. In Section 6, we demonstrate how our implementation of the synchronisation operation preserves strong fairness, privacy, and other properties of fair exchange. We use Section 7 to discuss some of the pending tasks that we have identified and placed on our agenda. Finally, in the conclusions Section 8, we share our implementation experience and the ideas generated from our work.

2. Related Work

The problem of fair exchange arises from the lack of mutual trust between parties. Asokan, Schunter, and Waidner (1997) proposed an exchange protocol for secure electronic payments using asymmetric cryptography, using a centralized TPP to coordinate the exchange [Asokan et al. 1997]. Later, Asokan, Shoup, and Waidner (1998) proposed optimistic fair exchange protocols that minimize TPP involvement, intervening only in cases of errors or disputes, while ensuring transaction security even against malicious actions [Asokan et al. 1998].

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 document under exchange. Our work appreciates the idea and progress 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. 1998] — our PBB is a simplified version his off–line TTP. We do not cover protocols with weak fairness such as gradual release (exchange); yet in passing we can mention that they are an alternative where TTPs are ruled out because they are not available, are too costly in comparison with the value of the items under exchange, introduce too much overhead or complexity. We can only suggest some references [Brickell et al. 1988, Pinkas 2003, Lan et al. 2007].

3. Background Information

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. 1998]. 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 FEP is fairness. From this perspective FEP can be divided into two classes:

- **Strong FEP:** 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 FEP:** guarantee only weak fairness, that is, they might produce unfair outcomes where one of the parties is left with the two items. These protocols account for potential occurrence of disputes and normally include (separated or embedded) dispute resolution mechanisms.

In 1999, Pagnia [Pagnia and Darmstadt 1999] demonstrated that fair exchange is at least as hard as consensus; therefore strong fairness can not be achieved without the involvement of a TTP. The TTP is a central controller used for keeping the invariant that the Finite State Machines (FSMs) of both participants reach the same final states. The salient feature of weak FEP is that they can be implemented without TTPs, unfortunately, they fall outside the scope of this paper (see Section 2).

3.1. Desirable properties of fair exchange

In addition to strong fairness, there are other properties that evaluate a protocol. We will mention only the most relevant ones here. 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. Anonymity requires that Alice and Bob do not disclose their identities to other parties, including TTPs. 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, anonymity is trivial to provide in gradual release; likewise, 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 cover only the first three properties. Additional information is in [Molina-Jimenez et al. 2024, Almutairi and Nigel 2019, Ray et al. 2005].

3.2. The five operations included in fair exchange

A close examination of fair exchange will reveal that the whole operation can be separated into five basic operations: handshake, deposit, verify, synchronisation, and release/restore [Molina-Jimenez et al. 2024].

We describe the operations from Alice's side; Bob's executions mirror Alice's. Handshake is executed by Alice and Bob to stipulate in a document the terms and conditions of the exchange, including the descriptions of their items. 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 she is expecting. synchronisation is performed to determine whether the exchange will take place. Finally, release is executed by a TTP to deliver Bob's item to Alice; otherwise, restore is executed to return Alice's item to Alice.

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 3.1) 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.

3.3. TTP architecture: monolithic *versus* split

A salient feature of existing strong FEP is that they implement their TTPs following a monolithic architecture (see Figure 1–a); typically, the TTP is a server with storage and computational facilities. As shown in the figure, the TTP is used for executing all the five basic operations (shown in bold.) discussed in Section 3.2. The appeal of such an architecture is its simplicity and familiarity; unfortunately, it compromises privacy.

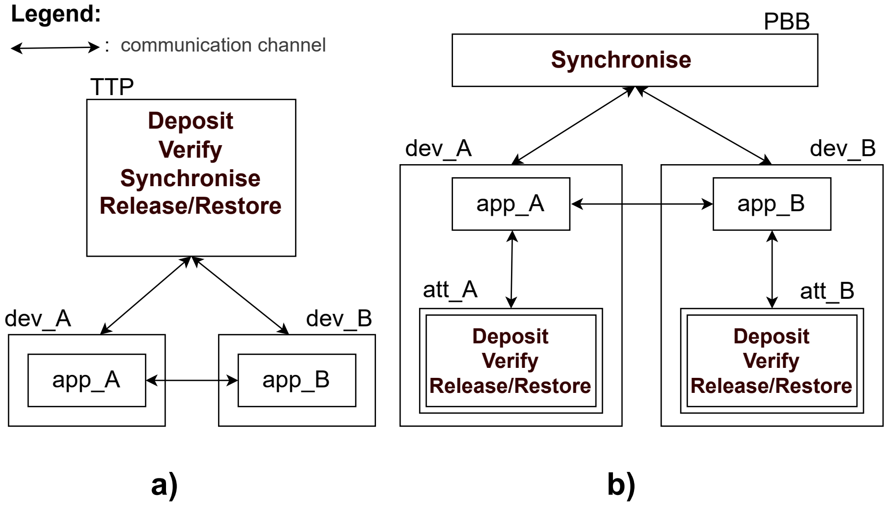

Fig. 1. a) Traditional monolithic TTP. b) Split TTP.

Another alternative is to use a split architecture shown in Figure 1–b. The TTP is composed of three components: two attestables (one for each participant) and a PBB. The attestables are trusted execution environments that comply with the attestable model discussed in Section 3.4. The attestable 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.). The main requirement 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, “oi”. The advantage of this approach is modularity. As shown in the Figure1–b, the execution the five 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 4.

3.4. The attestable model and current implementation technologies

An attestable is a model for executing code that manipulates sensitive data. It is expected to offers and 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 point. The model can be implemented using different hardware and software technologies. For example, we can use ARM TrustZone [Pinto and Santos 2019], Intel SGX [Mofrad et al. 2018], AMD SEV [Kaplan et al. 2016] and compartments created in Morello Boards [Grisenthwaite et al. 2023]. These technologies are becoming common in conventional mobile devices, servers and cloud platforms.

4. FEWD: a split TTP–based protocol that guarantees strong fairness and privacy

FEWD is a strong FEP that uses a split TTP. Its architecture is shown in Figure 2–a. A detailed description of the protocol can be found in [Molina-Jimenez et al. 2024]. Here we present only a summary with focus on the concepts needed to understand the implementation discussed in Section 5.

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 the figure, the attestable are shown in double lines to emphasise security protection and 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 the ideal scenario to explain the main ideas. Let us assume that Alice and Bob has conducted 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 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 $Sync\_B$: Alice's tokens, $Cancel\_A$ and $Cancel\_B$: Bob's tokens.

1. **deposit:** Alice and Bob deposit encrypted versions of their documents in the attestables of each other. As a result, $[D\_A]$ 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 abandons 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 recovers $Sync\_A$ and $Sync\_B$ from the log, it releases D_B to app_A. On the other side att_B will also recover $Sync\_A$ and $Sync\_B$ and release D_A to app_B.

In summary, documents are released through the execution of the `release` operation to their new owners applications (app_A and app_B) only when Alice post $Sync\_A$ and Bob posts $Sync\_B$. Documents not released remain locked for ever 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. Documents remain locked when Alice, Bob or both cancel the exchange. Figure 3 shows all possible developments of the protocol, including cancellations.

As shown in the figure, from the point of view of its functionality, FEWD is composed of two independent blocks: document manipulation and synchronisation exchange. The former is used for running the core of the protocol; as such, it keeps the state, manipulates the documents, and other sensitive information. The latter is used by Alice and Bob only to produce a binary outcome: 0 or 1 that in FEWD are interpreted as release or lock the documents. The subtlety is that this outcome can be produced without any knowledge of the core of the protocol. As demonstrated in Section 6, a separate and application–agnostic synchronisation enables privacy and other properties.

5. Implementation of the synchronisation operation

The specification of FEWD was published in 2024 [Molina-Jimenez et al. 2020] but without implementation. A complete implementation is time demanding. Luckily, the protocol can be cleanly separated into two parts (see Section 4) that can be implemented separately to scrutinise their properties. We are currently using Python to implement the synchronisation part to examine strong fairness and privacy.

Figure 2–b shows the modules of our current implementation. We have used TCP servers and clients that communicate with each other over secure channel created with SSL, to realise the components shown in Figure 2–a. 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 figure. 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 left 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.

To simplify the implementation, we have taken some diversions from the original specification [Molina-Jimenez et al. 2020].

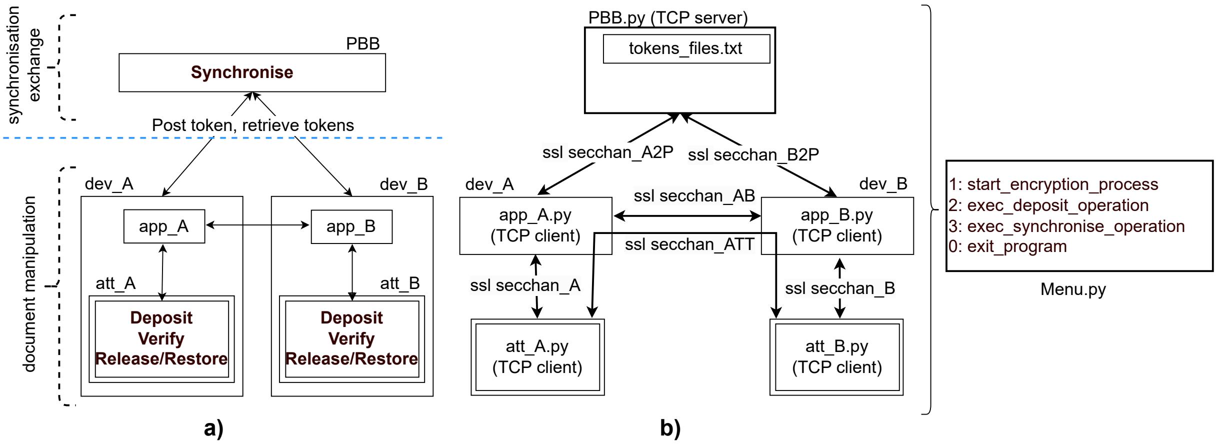

Fig. 2. a) FEWD with a split TTP. b) Python implementation.

1. In the original specification, the PBB stores tokens permanently. Our PBB keeps token only in memory, in a `txt` file.
2. In the original specification, the operations post and retrieve use independent communication sessions. Our implementation uses a single communication session is used for posting and retrieving.
3. In the original specifications, an attestable can post a token (e.g., $Sync\_A$) repeatedly and is responsible for scanning the retrieved log. Our implementation follows the same approach, allowing an attestable to post a token (e.g., $Sync\_A$) repeatedly, as stated in the original specifications [Molina-Jimenez et al. 2020]. However, while the original specifications require the PBB to return all received tokens, our implementation makes the PBB return only the tokens related to each specific exchange.

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

5.1. Implementation of the attestables

The attestables *att_A.py* and *att_B.py* are implemented as TCP clients. Incidentally, *att_A.py*, interacts with app_A, app_B and PBB. The *att_A.py* code can be divided into two parts. In part I (see Listing 1) Alice's attestable accepts a connection via a secure channel from app_A, receives a file D_A from app_A, and encrypts it.

In part II (see Listing 2) 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 user can press option "3" of the menu to execute the synchronise operation with instructions for att_A.*py* to post either $Sync\_A$ or $Cancel\_A$ to ***PBB.py***. After retrieving the tokens from the ***PBB.py***, att_A.*py* executes either release to surrender D_B to app_A or lock D_B for ever.

```
Accepts connection by a secure channel from app_A
Receives D_A from app_A
Encrypts the received file
```

Listing 1. Excerpt from Alice's attestable ($att\_A.py$**). Part I.**

```
Accepts connection by a secure channel from att_B through app_A.
Sends [D_A] to att_B through the channel.
Receives [D_B] from att_B through the channel.
Verifies [D_B]
Sends token to PBB (Sync_A or Cancel_A)
Releases/restores D_B to app_A
```

Listing 2. Excerpt from Alice's attestable ($att\_A.py$**). Part II.**

5.2. Implementation of the PBB

We have programmed the PBB (*PBB.py*) as a TCP server that remains listening for connections requests placed by the applications *app_A.py* and *app_B.py* to either post or retrieve tokens originally generated by *att_A.py* and *att_B.py*. It stores tokens in a list. Upon receiving a retrieve request, it returns the list. Listing 3 is an excerpt of *PBB.py*.

```
Receives tokens from app_A and app_B
Returns a set of tokens when a retrieve is requested.
```

Listing 3. PBB receiving syncronisation tokens originally posted by $att\_A.py$ $att\_B.py$**.**

5.3. Implementation of Alice's application

We have Alice's and Bob's applications (app_A.py and *app_B.py*, respectively) TCP clients. They 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.py* and *app_A.py* forwards it to the PBB. Listing 4 is an excerpt *app_A.py*. It shows the coding of the deposit operation to deposit D_A with *Att_B.py*.

```
Sends D_A to Alice's att_A
Forward items between Alice's att_A and Bob's att_B
Forward tokens between att_A and PBB
Receives D_B or a notification of abort.
```

Listing 4. Alice's application.

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 an abort notification.

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. We will use outputs from our execution to support our claims.

6.1. Strong fairness

The FSM of Figure 3 shows how strong fairness if guaranteed by our implementation. According [Molina-Jimenez et al. 2024], a strong fairness can only be guaranteed in protocols where the synchronisation 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 8 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.*

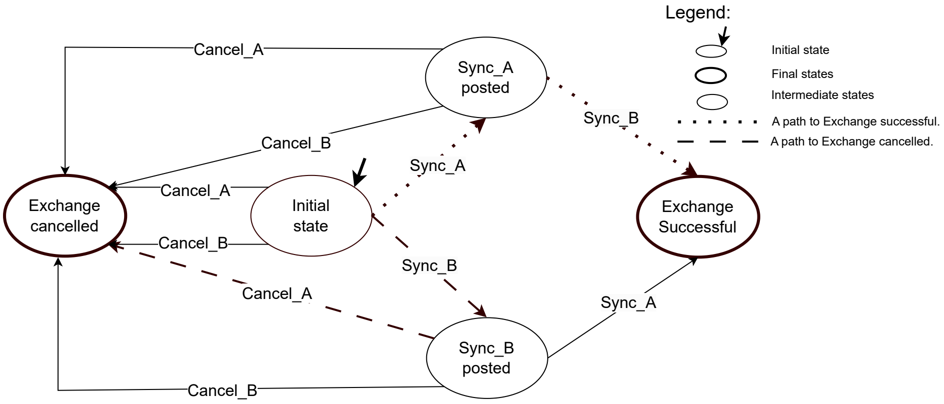

Fig. 3. FSM of sync operation with its two unique and mutually exclusive final states.

The Figure 3 shows one of the two paths that progresses from the initial to one of the final states, namely, *Exchange successful.* Also, the figure shows one of the six paths that progresses from the initial to the *Exchange cancelled* state, i.e., the final state where the exchange is cancelled.

The outputs produced from our implementation [1] when the path that leads to *Exchange successful* develops are shown in Listing 5.

```
Alice sent: Sync_A to PBB /*S_A posted state reached
Bob sent: Sync_B to PBB /*S_A and S_B posted state reached
Got a connection from ("127.0.0.1", 54548)
The PBB Received the Sync_A message from Alice's token
The PBB Received the Sync_B message from Bob's token
The PBB responded with: Sync_A, Sync_B to Alice and Bob
Alice's attestable sends D_B to your application.
Bob's attestable sends D_A to your application.
```

Listing 5. Execution outputs: PBB synchronises to release the documents.

The outputs produced from our implementation when the path that leads to *Exchange cancelled* develops are shown in Listing 6.

```
Bob sent: Sync_B to PBB
Alice sent: Cancel_A to PBB
Got a connection from ('127.0.0.1', 55810)
The PBB Received the Sync_B message from Bob's token
The PBB Received the Cancel_A message from Alice's token
The PBB responded the attestable with: Sync_B, Cancel_A to Alice and Bob
Alice's attestable sends notification of abort to your application
Bob's attestable sends notification of abort to your application
```

Listing 6. Execution outputs: PBB synchronises to cancel the exchange (no release).

6.2. Privacy

In SFEP privacy in put at risk by the information that Alice and Bob reveal to the TTP. As explained in Section 3, the latter is needed at least for the execution of the synchronisation operation [Pagnia and Darmstadt 1999]. Figure 3 shows how our implementation can execute the synchronise operation without revealing sensitive information to the PBB: Observer 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 *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 cancelled from the PBB or from other parties that might have access to it. With this approach, the PBB is application–agnostic and stateless. It is only a token repository that offers two services: it stores tokens and delivers tokens.

In the original description of FEWD [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

[1] We have edited the outputs and added the explanatory lines marked with verb+/*+.

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 attestables can post a token (e.g., $Sync\_A$) repeatedly. The PBB is responsible for receiving tokens and recording them in a `txt` file that is associated to each exchange. The association of tokens to instances of the protocol simplifies the task of filtering tokens in the PBB and therefore simplifies the implementation of the PBB. It also frees the attestables from the task of token filtering. The only reservation is that the PBB receives an ID attached to the tokens that identifies the protocol instance. We admit that this implementation of PBB reveals some data about the exchange, but it does not reveal any data about the items.

6.3. Strong timeliness

Timeliness is provided by the freedom that Alice and Bob have to post cancellation tokens. Figure 3 shows the state transitions driven by posted tokens. Figure 4 shows the four paths that can develop when Alice posts firstly. The remaining four paths that develop when Bob posts firstly are similar and omitted. The symbol "*" means any token posted by Alice or Bob or no token posted. We say that a token is irrelevant if it has no effect in reaching the final outcome: either *Exchange cancelled* or *Exchange successful*.

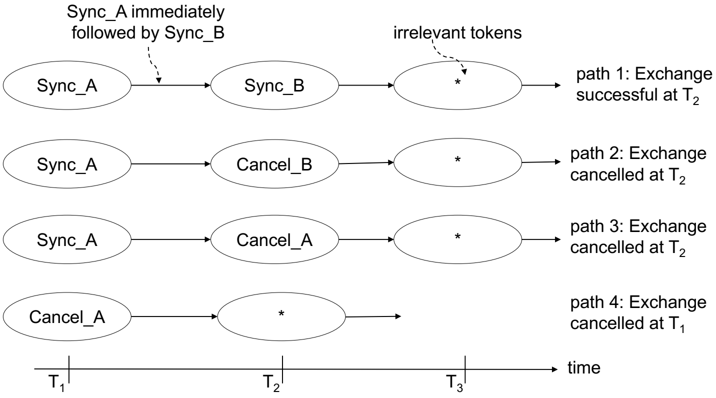
Sync_A immediately followed by Sync_B
irrelevant tokens
Sync_A
Sync_B
*
path 1: Exchange successful at T2
Sync_A
Cancel_B
*
path 2: Exchange cancelled at T2
Sync_A
Cancel_A
*
path 3: Exchange cancelled at T2
Cancel_A
*
path 4: Exchange cancelled at T1
T1
T2
T3
time

Fig. 4. Post of cancel tokens to guarantee timeliness at synchronisation.

Figure 4, shows that Alice can use token $Cancel\_A$ to cancel, admittedly, under certain restrictions that materialise when she changes her mind. 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 the protocol 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 $Cancel\_B$ kills her plans. The execution output shown in Listing 7 shows the development of path 2 of Figure 4.

```
Alice sent: Sync_A to PBB
Bob sent: Cancel_B to PBB
Got a connection from ('127.0.0.1', 61925)
The PBB Received the Sync_A message from Alice with your token
The PBB Received the Cancel_B message from Bob with your token
The PBB responded with: Sync_A, Cancel_B
Alice's attestable sends notification of abort to your application
Bob's attestable sends notification of abort to your application
```

Listing 7. Execution outputs: Bob takes advantage of timeliness—he cancels early.

Figure 5–a shows the timelines of path 2 of Figure 4. The interval before posting $Sync\_A$ at T_1 is safe for Alice, i.e., she is not at risk of loosing her item. T_1–T_6 is risky for Alice: she cannot cancel the protocol categorically or abandon it. She needs to waits till 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 5–b shows path 3 where Alice is safe before posting $Sync\_A$ at T_1; she enters a risky interval after 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.

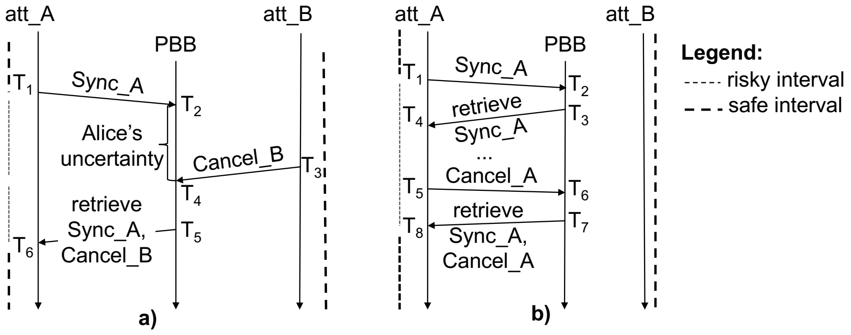

Fig. 5. Timeliness with safe, uncertain and risky intervals.

7. Future work

We have implementation and fundamental research tasks pending. The code available from GitHub reveals that our implementation of the FEWD protocol is an early stage. In our agenda is the replacement of the Python code that we use to play the role of the attestable by actual technology. It is very likely that the latter uncover several engineering challenges and performance issues related to cryptography. A more fundamental research question is the examination of the synchronisation operation in FEWD. In Figure 2–a it is implemented with the help of the PBB, as a results it guarantees strong fairness and privacy; yet we observed that other synchronisation mechanisms can be used to release or restore the items.

Accordingly, we are planning to replace the PBB–based synchronisation by other synchronisation protocols and compare their performance. In particular, we will implement and use protocols that guarantee only weak fairness like gradual release and the Keep in Touch protocol discussed in [Avoine and Vaudenay 2004]. As explained in the original description of FEWD [Molina-Jimenez et al. 2020], fair exchange rises 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?

8. 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 trusted execution environment simplifies the fair exchange problem (and other similar problems) and can help in building simpler protocols with new properties. The Python implementation discussed here consists of about 1.470 lines. Yet, the protocols have to be implemented and their properties tested for practicability. Our results can defend strong fairness, privacy and timeliness. Yet, we can ask if the items left forever locked in the attestable when the protocol is cancelled, are inaccessible for ever or only temporary safe, that is till Bob hacks his attestable and extracts Alice's document? There are other questions that we did not address. For example, to what extent the description of items used in verify give away the items? From our implementation experience, this questions seems harder than synchronisation.

Availability of artifacts

The source code for the implementation of the synchronisation protocol for fair exchange with strong fairness and privacy is available at the following URL: `https://github.com/gca-research-group/fair-exchange`

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) grant funded Carlos Molina-Jimenez and Jon Crowcroft.

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.

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 Waider., M. (1998). Asynchronous protocols for optimistic fair exchange. In *Proceedings Symposium on Security and Privacy*, pages 86–99.

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.

Avoine, G. and Vaudenay, S. (2004). Fair exchange with guardian angels. In *Proceedings Information Security Applications: 4th International Workshop*, pages 188–202.

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.

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.

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

Lan, T., Qin, Z., Zhao, Y., Xiong, H., and Liu, L. (2007). A gradual and optimistic fair exchange protocol. In *Proceedings International Conference on Communications, Circuits and Systems*, pages 452–456.

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

Mofrad, S., Zhang, F., Lu, S., and Shi, W. (2018). A comparison study of intel sgx and amd memory encryption technology. In *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*, pages 1–8.

Molina-Jimenez, C., Nakib, H. D. A., Song, L., Sfyrakis, and Crowcroft, J. (2020). A case for a currencyless economy based on bartering with smart contracts. *arXiv preprint arXiv:2010.07013*.

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 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.

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.