28 December 2020 Study · Transparency · 21 min read

Correctness, Completeness, and Non-Repudiation of Transparency Logs

Archived. This content is no longer updated. It remains available for reference.

Source https://technology2.a-sit.at/en/publications/correctness-completeness-and-non-repudiation-of-transparency-logs/

Correctness and Completeness of are two desirable features for Transparency Logs. Unfortunately, when it comes to data processing events, these features cannot be guaranteed due to lack of mechanisms to prevent data processors from logging incorrect information or not recording a data processing event in the log. This project aimed at investigating mechanisms to overcome this challenge and to ensure non-repudiation of data transactions. Potential non-repudiation protocols such as fair exchange protocols were analyzed in connection with existing logging mechanisms.

1. Introduction

Data Protection legislations such as the GDPR encourage the creation and the maintenance of a correct and complete Transparency Log of private data access and processing events. In this context, we define the desired Completeness, Correctness, and Non-repudiation properties of data transparency logs as follows:

Completeness: *All* data processing transactions should be logged in Transparency Logs. It should not be possible to omit to record a data processing event. *Correctness*: The log content should be correct with respect to the real data processing transactions. *Non-repudiation*: It should not be possible to deny later in time that a specific data processing event took place in the past.

To this day, besides voluntary compliance, there is no technical means to ensure the correctness and completeness of transparency logs, as it is almost impossible to prevent a data processor from entering false data or omitting to log a certain event. Hence, it is impossible to fully achieve correctness and completeness for data processing operations where only one party, (namely the data processor) is involved. On the other hand, for operations where two or more (non-colluding) parties are involved the situation is different. Typically, these operations involve consent release or data transfer events from the data subject to the data processor. In such scenarios, data processors feel the need for technical means to ensure the creation and maintenance of a correct and complete Transparency Log. Data Processors can further exploit such ledgers to prove compliance to regulations to both data subjects and data protection

authorities. Further incentives to correctness involve signing of the event record by the parties involved in a data access operation. The creation of such "joint" or "simultaneous" signatures has been well researched in the past by fair exchange protocols [1]. The objective of fair exchange protocols is to ensure, as the name suggests, fair exchange of electronic data (digital signatures, payment or confidential data) between two or more parties. A protocol is considered to be *fair* if at the end of it, either all participants receive the expected item, or none of them does. Such protocols have found applications in contexts such as certified e-mail delivery, consumer card payments, or delivery of invoices. This project explores the viability of employing such protocols in connection to secure logging mechanisms in order to ensure non-repudiation of data transactions. The parties involved the data sharing transactions will mutually generate digital evidence that classifies as proof of non-repudiation of receipt. Such an approach can be used to provide certain guarantees that the personal data transfer operations can be completed only if all the participating parties sign the log record and the record is added to the log, and hence improve the completeness, correctness, and non-repudiation of personal data sharing events.

2. Fair Exchange Protocols

This section reviews the main methods for fair exchange protocols, as a promising solution to improve the correctness and completeness of transparency logs. We first introduce some basic terminology and continue to further survey the existing algorithms and their properties.

2.1. Basic notions and properties

In the simplified two party fair exchange setting, two parties A and B need to exchange two objects OA and OB, such that A receives OB if and only if B receives OA. This setting can further be extended to multi-party fair exchange, where multiple parties interact to exchange such objects. Due to their generic nature, fair exchange protocols can be instantiated for different scenarios as a solution to different problems. Examples include digital contract signing, certified e-mail, and ecommerce. Each instantiation context may pose specific requirements. For data sharing events, the fair exchange protocol can be instantiated as follows:

The involved parties (data subjects, data processors) involved in the exchange require the following guarantees:

i. The receiving party wants to receive the object of interest (data subject's consent, or the shared data) together with a proof of origin, to ensure non-repudiation; ii. The sending party requires a receipt, again for non-repudiation, as a proof that the recipient has received the object of interest. Alternatively, the receipt can also be proof that the data sharing transaction was recorded in a transparency log, ensuring in this case, the correctness and the completeness of the transparency log.

This section lists the main parameters, which are used in literature to classify fair exchange algorithms. We illustrate the general features of fair exchange methods and discuss about their applicability to the transparency log scenario.

2.1.1. Trusted Third Party Involvement

Fair exchange protocols are widely classified as *online* / *inline* and *offline* (or *optimistic*) protocols according to the involvement of a Trusted Third Party (TTP):

Inline TTP Protocols : This category of fair-exchange protocols involves a TTP for every data transmission. Hence, each exchanged message is either sent by an involved party to a TTP, or sent by the TTP to another party.

Online TTP Protocols : These protocols involve a TTP for each protocol execution, but not necessarily for each data transmission step, i.e. for every run of the protocol, there exists at least one message exchange that goes through the TTP, while other direct message exchanges between parties are possible.

Offline TTP Protocols : Also called as optimistic protocols, these protocols involve a TTP only when one of the involved parties misbehaves, while in the other cases, the parties run the protocol by themselves.

Transparent TTP Protocols: During the execution of transparent TTP protocols, the TTP generates message exchange evidences that are indistinguishable from the evidences produced by the other parties. Thus, just by looking at the evidences generated at the end of the protocol, one cannot tell whether the TTP was involved or not.

Protocols without TTP typically provide weaker (probabilistic) guarantees, or rely on assumptions with respect to the computational power of the parties involved. Therefore, they are considered out of scope for this project. Inline TTP protocols are usually simpler, but have the imminent risk of the TTP becoming a bottleneck. Offline or optimistic protocols have advantages when compared to online/inline TTP protocols as they rely on the assumption that the TTP intervention is not needed for the majority of cases. These protocols typically first describe the algorithm for exchange with honest parties, and then describe the recovery procedures that the parties or the TTP have to follow in case there is a failure or timeout in the main algorithm steps.

2.1.2. Fairness Concept Definitions

Since the introduction for fair exchange protocols, there have been different approaches to the fairness concept, such as computational or probabilistic fairness. Such definitions of fairness typically imply a multi-step protocol, in which the parties involved gradually disclose the information to be exchanged. The general definition of fairness considers the protocol to be atomic, i.e. either all of its sub-actions are executed, or none at all. The different fairness concepts are defined in literature as follows:

Computational Fairness: This definition of fairness has at its core the strong assumption of equal resource assumption, i.e. computational fairness assumes that the computational capabilities of the parties involved are (approximately) equal. [2] *Probabilistic Fairness:* This fairness definition relaxes the strong assumption of equal resource consumption by adding probabilistic arguments. Such protocols define fairness in the sense that, at any stage of the execution of the protocol, the probability of one party being slightly advantaged with respect to the other party is kept within acceptable limits, and close to zero. [3] *General Definition of Fairness:* The most widely generally accepted fairness definition in the literature is as follows: At the end of execution of a fair exchange protocol, either all involved parties receive their expected information of none of them does. [4]

2.1.3. Communication Channel

Protocols proposed for fair-exchange of items generally use one of the following channels for communication:

Unreliable Channel: Communication channels that provide no authentication mechanism or guarantees of data transmission whatsoever.

Resilient Channel: A resilient channel provides guarantees of transmitting the data correctly after a finite, but unknown, period of time, but does not guarantee the correct order of transmission.

Operational Channel: An operational channel provides guarantees of correct data transmission which a pre-defined period of time.

2.1.4. Protocol Properties

As aforementioned, different instantiations of the fair exchange problem may pose different requirements (or desired properties) based on their specific context. The most common properties include but are not limited to: *Fairness*, *Non-repudiation (of origin and receipt),* and *Authenticity, Integrity, and Confidentiality (of the messages exchanged)* . Furthermore, some instantiations may pose time requirements, such as Timeliness (the protocol finished within a pre-defined time period) and Temporal Authentication (such as timestamps). These requirements apply to the context of personal data sharing as well and must be considered when designing a fair exchange protocol for personal data access transparency.

2.2. Protocols Review

This section reviews some of the main fair-exchange protocols that have been proposed for both the two-party and the multi-party setting.

2.2.1. Two-party Protocols

Online TTP Protocols Protocols with online TTP involve the TTP in every execution of the protocol to guarantee fairness and accountability. This means that for every protocol run, there exists at least one exchange that takes place through the TTP. Abadi et al. in [5] or Cox et al. with the NetBill protocol [6] proposed an online TTP protocol for certified e-mail systems and digital goods exchange, respectively. The proposed online TTP solutions involve the TTP in different roles. In the protocol by Abadi et.al. [5] the TTP is responsible for forwarding the encrypted e-mails and the keys used to encrypt them to the receiving party, and the receipts to the sending party. The TTP in the proposal by Rabin for transaction protection by beacons [7] is used to broadcast messages that contain public/private keys in each round of the protocol execution. Other protocols, such as the one proposed by Zhang in [8] use the TTP to properly manage and then push the correct encryption keys to the parties involved.

Inline TTP Protocols Inline TTP Protocols involve the TTP in every message exchange. Although these protocols are generally simpler and provide greater accountability, the TTP may often become a computation or communication bottleneck. Examples of solutions proposed under this category include the protocol by Bahreman and Tygar in [9] for certified electronic mail, or the non-repudiation protocols proposed by Zhou and Gollmann in [10] and Coffey and Saidha in [11]. Moreover, protocols such as [11] may require multiple TTPs for different purposes, such as a dedicated Timestamping TTP and a dedicated Non-Repudiation TTP.

Offline TTP Protocols Offline TTP Protocols are called optimistic as they assume that the parties will follow the protocol honestly, and the intervention of the TTP will not be needed for most of the time. The degree of fairness of these protocols is further described in terms of two additional properties, namely the *revocability* (whether the third party is able to undo a transaction) of the items, and the so-called *generatability* of them (if the third party is able to generate a replacement for them) Asokan et al. [12] were among the first ones to introduce an offline TTP protocol for the fair exchange of generic items. They also introduced the idea of revocable items in the presence of operational channels. According to the authors, strong fairness is achievable if at least one of the exchanged items is either revocable or generatable / replaceable from the TTP. Later in time, Asokan et al. in

[13] and Zhou and Gollmann in [10] proposed a protocol that provides strong fairness guarantees in the presence of resilient channels and unreliable channels, respectively.

Transparent TTP Protocols In the cases of offline TTP protocol executions where TTP intervention is required, the third party typically generates evidences that are identical to the evidences produced by either the sending or receiving party in an honest protocol flow. The evidence generated by such transparent or invisible TTPs is such that, it is impossible to tell whether the TTP was involved in the protocol execution or not, just by looking at these evidences. Micali [14] was the first author to propose the usage of an invisible TTP for the certified e-mail instantiation of the fair exchange problem. Furthermore, Asokan et al. in [15], Ateniese in [16] and Bao et al. in [17] proposed solutions based on verifiable convertible signatures and verifiable encryption, respectively, that allowed to recover the parties' signatures for evidence creation rather than having the TTP sign them.

2.2.2. Multi-party Protocols

Multi-party fair-exchange protocols have been proposed for contexts where multiple parties need to exchange items fairly. These protocols can mainly be classified based on the *topology* of the graph that represents them, with the nodes of the graph representing the vertices of the graph, and each exchange between them being and edge. Franklin and Tsudik [18] and Bao et al. [19] proposed solutions that were mainly based on the ring topology. Asokan et al. [20] proposed multi-party fair-exchange protocols on synchronous networks for the more general matrix topology, where each entity receives items from or offers items to a set of entities. Another proposed topology more suited for non-repudiation fair protocols is the star topology, where each entity sends a message to more than one entity and they respond to the sender. Examples of such solutions include the approach from Kremer and Markowitch in both [4] and [21] which define multi-party non-repudiation fair protocols with both online and offline TTPs. Both of these solutions achieve significant performance increase when compared to n two-party protocols.

3. Protocol Design and Implementation

Only a small fraction of the reviewed fair exchange methods are applicable to the personal data sharing context. Throughout this project, an optimistic fair exchange protocol was designed/extended and then implemented based on the basic protocol first proposed by Micali [14], and a variant of it further proposed by Oniz et al. in [22].

3.1. Optimistic Fair Exchange Basic Protocol

The idea of optimistic fair exchange protocols was introduced in [17], [13], and [12] for the exchange of generic items, and in [14] for the exchange of certified mail. The latter is also the protocol that we use as a base for the design of an optimistic fair exchange protocol for data sharing.

The main steps of the algorithm with honest parties involved are as follows:

i. $A \rightarrow B: Z$, $Z = ETTPR\ (A||B||M)$ ii. $B \rightarrow A: [Z]B$ iii. $A \rightarrow B: (M||R)$ In the first step, Alice (A) sends Bob (B) Z: the encryption under the TTP's public key of the concatenation of Alice's identifier, Bob's identifier, and the message M. Upon receiving Z from Alice, Bob signs it and sends [Z]B to Alice as a receipt. In the third step, if Alice receives the correctly signed receipt from Bob, she sends the plaintext message M and the random number R to Bob. At the end of the execution of the protocol, Bob has the message M, and Alice has the receipt [Z]B.

Considering that for the context of Transparency Logs, the confidentiality of the message M to be exchanged is a crucial requirement, the message M is replaced by EB(M) across the steps of the algorithm. By doing so, no other party besides B, not even the TTP, is able to extract the plaintext message M.

In case one of the parties behaves dishonestly, or a time-out is reached and one of the aforementioned steps fails, the protocol continues as follows:

- If B fails to send back the receipt [Z]B, then A consequently does not send back the message M and the protocol executions stops., Fairness is guaranteed in the sense that none of the parties get what they required.
- If B sends back the receipt, but A does not follow with step 3, B contacts the TTP and sends the message Z and the receipt ZB. The TTP extracts M and R by using its private key and sends them to B. Finally the TTP sends the receipt [Z]B:

iv. $B \rightarrow TTP: Z||[Z]B$ v. $TTP \rightarrow B: (M||R)$ vi. $TTP \rightarrow A: [Z]B$

- Furthermore, in case a mismatch exists between the messages sent in each step, i.e. A sends a M′ and R′ such that B computes E′≠E, the same steps (iv,v,vi) are executed. The protocol assumes that at least the communication channel between the parties and the TTP is resilient.

3.2. Optimistic Fair Exchange Protocol for Data Sharing

We adapt the protocol depicted in Section 2.4 to meet the requirements of the data sharing instantiation of the problem as follows:

i. A generates a session key k, to be used to encrypt the private dataset D before sending it to B. ii. $A \rightarrow B: Z1$, $Z1 = [A \,||\, B \,||\, Ek(D) \,||\, h(Ek(D)) \,||\, t0\,]A$ iii. $B \rightarrow A: [Z1]B$ iv. $A \rightarrow B: Z2$, $Z2 = [A \,||\, B \,||\, Ek(D) \,||\, ETTP(k) \,||\, t1\,]A$ v. $B \rightarrow A: [Z2]B$ vi. $A \rightarrow B: [\,EB(k)]A$ In the second step A sends the first message Z1 to B, which consists of the concatenation of the identifiers of A and B, the encrypted dataset, its hash value, and a unique timestamp t0. Upon receiving Z1, B checks whether the hash value of the encrypted dataset, and the dataset sent matches, and either sends the receipt [Z1]B to A, or aborts the protocol otherwise. Upon

receiving the receipt for the first message, A sends B a second message Z2, which contains the session key encrypted under the TTP's public key. B sends a second receipt for Z2, and A finally sends the session key k encrypted using B's public key. B decrypts the dataset using k and verifies the result.

In case any of the parties misbehaves, or a failure occurs, the recovery protocol involving the TTP is defined as follows:

vii. $B \rightarrow TTP : [Z2]B$ viii. $TTP \rightarrow B : [\ EB(k)]TTP$ ix. $TTP \rightarrow A : [[Z2]B]TTP$ In case the protocol runs with no failures, at the end of it, B has the dataset of A, and A has the two receipts [Z1]B and [Z2]B. In case a failure occurs before step v, the protocol is aborted, B has no possibility to decrypt D. In case a failure occurs during step v or vi, the TTP recovery protocol is run, and both parties get their required items. The dataset is kept confidential and B is the only party that can decrypt it, given that he receives EB(k). The fact that each message is signed by the parties involved, ensures non-repudiation of origin. The unique timestamps and the generated session key serve the purpose of addressing replay attacks.

4. Application Run-Through

This section illustrates some of the main execution paths of the fair exchange protocol that was implemented within this project. During the initial setup, key-pairs need to be generated for the sending (A), receiving (B), and Trusted Third Party (TTP).

```
[redacted]-Virtual-Machine:~/dev/ofepj$ ./run -gk TTP && ./run -gk A && ./run -gk B
Saved the public key in file TTP-public.key
Saved the private key in file TTP-private.key
Keys generated with success!
Saved the public key in file A-public.key
Saved the private key in file A-private.key
Keys generated with success!
Saved the public key in file B-public.key
Saved the private key in file B-private.key
Keys generated with success!
```

Figure 1: Initial Setup - Key Generation Each of the parties is run in a different terminal window. The corresponding addresses of the parties to communicate to, as well as the relevant key material are given as input arguments. Figure 2 illustrates an example execution with honest parties. Initially, A is prompted to enter the message to be exchanged. A sends the message encrypted using a random session key, and waits for B's signed receipt. Upon receiving the encrypted message, B sends back the signed receipt. A checks that the message and the signature matches and verifies correctly, and if so sends a second message that contains the key K to B. B uses this key to decrypt and extract the message. The protocol execution ends without the need of the intervention of the TTP.

```
[redacted]-Virtual-Machine:~/dev/ofepj$ ./run -d -a -ai A -bi B -ba localhost:5002 -ta localhost:5001
Debug mode enabled.
Type the message: This is the consent for my private data.
Starting in mode A. Connected to localhost on port 5002
Starting step 0.
Ended step 0.
To abort now type 'a'. To continue type anything else.

Waiting for an object.
Received object of type java.security.SignedObject
Starting step 1 - want a SignedObject.
EOOm matches? true
Signature matches? true
EORm validation succeeded!
To resolve now type 'r'. To send an invalid K press 'k'. To send a bogus message press 'b'. To continue type anything else.

Ended step 1.
Waiting for an object.
Received object of type java.security.SignedObject
Starting step 2 - want a SignedObject.
Urray! Mission accomplished.
Ended step 3.
```

Figure 2: Protocol Execution with honest parties We continue with alternative execution flows that require the recovery procedure defined for the TTP. An example of such a scenario is when B sends back a receipt signed with a wrong key. In that case, the signature validation performed by A on B's receipt fails, and A contacts to TTP to notify B's misbehavior and aborts the protocol execution. Figure 3 depicts this execution flow.

```
[redacted]-Virtual-Machine:~/dev/ofepj$ ./run -d -b -bi B -lp 5002 -ta localhost:5001
Debug mode enabled.
Starting in mode B. Listening on port 5002
Someone connected.
Waiting for an object.
Received object of type protocols.exchange.Hello
Starting step 0 - want a Hello
Received a Hello from the user with identity A
Ended step 0.
Waiting for an object.
Received object of type javax.crypto.SealedObject
Starting step 1 - want a SealedObject.
Ended step 1.
Waiting for an object.
Received object of type java.security.SignedObject
Starting step 2 - want a SignedObject.
EOOm verification succeeded!
To resolve now type 'r'. To sign the EORm with a bogus key press 'k'. To send a bogus message press 'm'. To continue type anything else.

Ended step 2.
Waiting for an object.
Received object of type protocols.exchange.K
Starting step 3 - want a K
Urray! The exchanged message is: This is the consent for my private data.
To resolve now type 'r'. To fake the EORk press 'k'. To continue type anything else.

Ended step 3.
```

```
[redacted]-Virtual-Machine:~/dev/ofepj$ ./run -d -a -ai A -bi B -ba localhost:5002 -ta localhost:5001
Debug mode enabled.
Type the message: This is the consent for my private data.
Starting in mode A. Connected to localhost on port 5002
Starting step 0.
Ended step 0.
To abort now type 'a'. To continue type anything else.

Waiting for an object.
Received object of type java.security.SignedObject
Starting step 1 - want a SignedObject.
EOOm matches? true
Signature matches? true
EORm validation succeeded!
To resolve now type 'r'. To send an invalid K press 'k'. To send a bogus message press 'b'. To continue type anything else.

Ended step 1.
Waiting for an object.
Received object of type java.security.SignedObject
Starting step 2 - want a SignedObject.
Urray! Mission accomplished.
Ended step 3.
```

```
[redacted]-Virtual-Machine:~/dev/ofepj$ ./run -d -b -bi B -lp 5002 -ta localhost:5001
Debug mode enabled.
Starting in mode B. Listening on port 5002
Someone connected.
Waiting for an object.
Received object of type protocols.exchange.Hello
Starting step 0 - want a Hello
Received a Hello from the user with identity A
Ended step 0.
Waiting for an object.
Received object of type javax.crypto.SealedObject
Starting step 1 - want a SealedObject.
Ended step 1.
Waiting for an object.
Received object of type java.security.SignedObject
Starting step 2 - want a SignedObject.
EOOm verification succeeded!
To resolve now type 'r'. To sign the EORm with a bogus key press 'k'. To send a bogus message press 'm'. To continue type anything else.

Ended step 2.
Waiting for an object.
Received object of type protocols.exchange.K
Starting step 3 - want a K
Urray! The exchanged message is: This is the consent for my private data.
To resolve now type 'r'. To fake the EORk press 'k'. To continue type anything else.

Ended step 3.
```

```
[redacted]-Virtual-Machine:~/dev/ofepj$ ./run -d -ttp -lp 5001
Debug mode enabled.
```

```
[redacted]-Virtual-Machine:~/dev/ofepj$ ./run -d -a -ai A -bi B -ba localhost:5002 -ta localhost:500
Debug mode enabled.
Type the message: This is my private data.
Starting in mode A. Connected to localhost on port 5002
Starting step 0.
Ended step 0.
To abort now type 'a'. To continue type anything else.

Waiting for an object.
Received object of type java.security.SignedObject
Starting step 1 - want a SignedObject.
EOOm matches? true
Signature matches? false
EORm validation failed.
Connected to the trusted third-party. Starting the abort process.
Session aborted.
Quitting.
```

```
[redacted]Virtual-Machine:~/dev/ofepj$ ./run -d -b -bi B -lp 5002 -ta localhost:5001
Debug mode enabled.
Starting in mode B. Listening on port 5002
Someone connected.
Waiting for an object.
Received object of type protocols.exchange.Hello
Starting step 0 - want a Hello
Received a Hello from the user with identity A
Ended step 0.
Waiting for an object.
Received object of type javax.crypto.SealedObject
Starting step 1 - want a SealedObject.
Ended step 1.
Waiting for an object.
Received object of type java.security.SignedObject
Starting step 2 - want a SignedObject.
EOOm verification succeeded!
To resolve now type 'r'. To sign the EORm with a bogus key press 'k'. To send a bogus message press 'm'. To continue type anything else.
k
Ended step 2.
Waiting for an object.
Connection timed-out.
Connected to the trusted third-party. Starting the resolution process.
Session was already aborted by the other party. Unable to resolve this session.
Quitting.
```

Figure 3: Protocol Execution with TTP intervention – incorrect signature key We further consider cases where the sending party, namely A, strays away from the honest protocol and sends a bogus message as the second message, after receiving a correct receipt from the receiving party B. In this case, B contacts the TTP with the first message that A sent, and the receipt sent by him. The TTP decrypts the message (using its private key) and sends B the key K that enables him to decrypt the message sent in step 1 by A. This execution flow is shown in Figure 4.

```
[5] New connection from /127.0.0.1:45564
[5] Waiting for an object.
[5] Received object of type java.security.SignedObject
[5] Resolving.
[5] Sending ABt
```

```
[redacted]Virtual-Machine:~/dev/ofepj$ ./run -d -b -bi B -lp 5002 -ta localhost:5001
Debug mode enabled.
Starting in mode B. Listening on port 5002
Someone connected.
Waiting for an object.
Received object of type protocols.exchange.Hello
Starting step 0 - want a Hello
Received a Hello from the user with identity A
Ended step 0.
Waiting for an object.
Received object of type javax.crypto.SealedObject
Starting step 1 - want a SealedObject.
Ended step 1.
Waiting for an object.
Received object of type java.security.SignedObject
Starting step 2 - want a SignedObject.
EOOm verification succeeded!
To resolve now type 'r'. To sign the EORm with a bogus key press 'k'. To send a bogus message press 'm'. To continue type anything else.
k
Ended step 2.
Waiting for an object.
Connection timed-out.
Connected to the trusted third-party. Starting the resolution process.
Session was already aborted by the other party. Unable to resolve this session.
Quitting.
```

```
[5] New connection from /127.0.0.1:45564
[5] Waiting for an object.
[5] Received object of type java.security.SignedObject
[5] Resolving.
[5] Sending ABt
```

```
[redacted]-Virtual-Machine:~/dev/ofepj$ ./run -d -a -ai A -bi B -ba localhost:5003 -ta localhost:5002
Debug mode enabled.
Type the message: This is my consent.
Starting in mode A. Connected to localhost on port 5003
Starting step 0.
Ended step 0.
To abort now type 'a'. To continue type anything else.

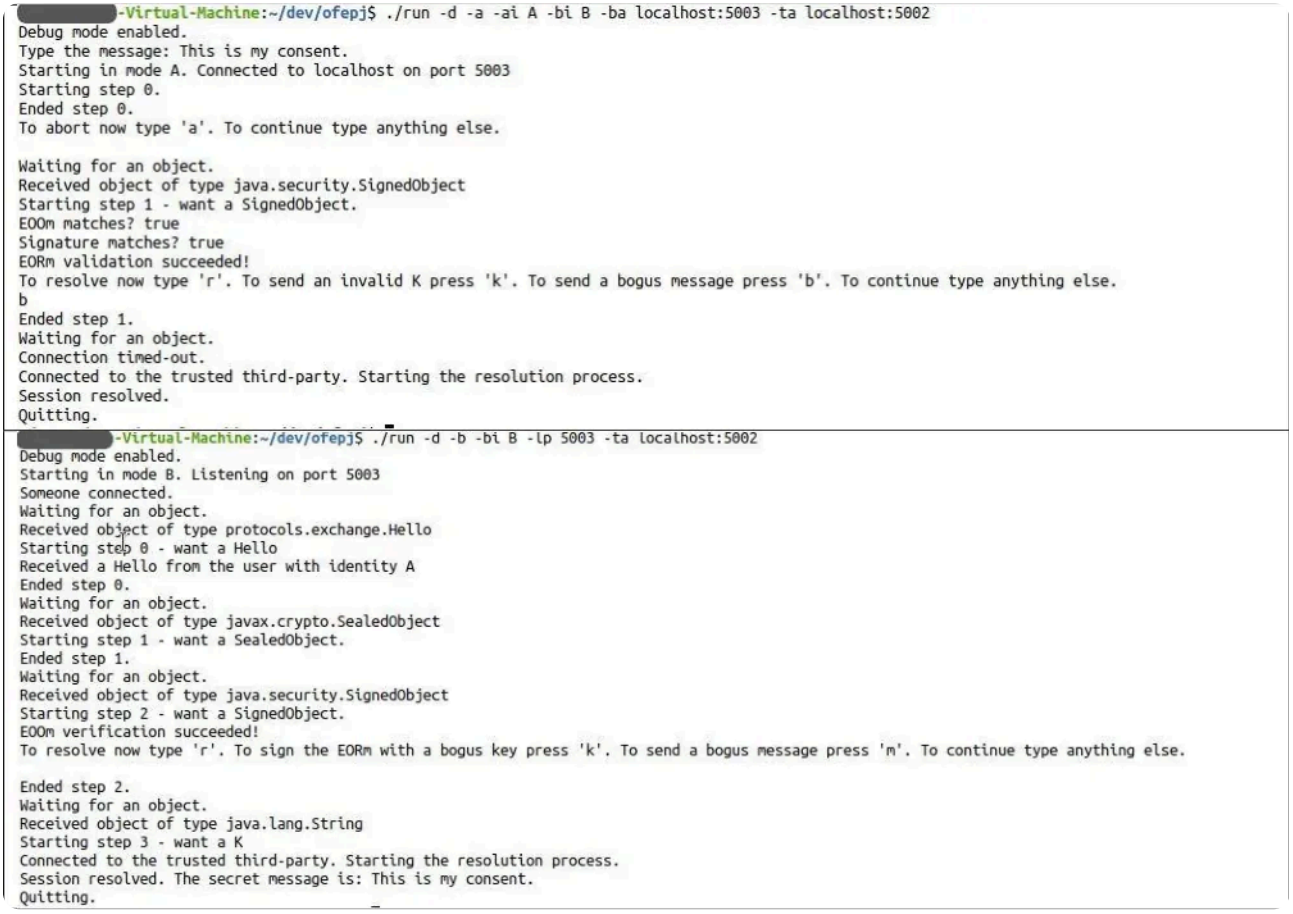
Waiting for an object.
Received object of type java.security.SignedObject
Starting step 1 - want a SignedObject.
EOOm matches? true
Signature matches? true
EORm validation succeeded!
To resolve now type 'r'. To send an invalid K press 'k'. To send a bogus message press 'b'. To continue type anything else.
b
Ended step 1.
Waiting for an object.
Connection timed-out.
Connected to the trusted third-party. Starting the resolution process.
Session resolved.
Quitting.
```

Figure 4: Protocol Execution with TTP intervention - A sends bogus second message A might also misbehave and send an incorrect key K rather the one that was used to encrypt the message in the first place. In this case, the execution flow is similar to the previous one. B contacts the TTP, which follows the recovery procedure. At the end of the protocol, B is able to extract and decrypt the message thanks to TTP's intervention, as illustrated in Figure 5. Consequently, fairness is guaranteed for each of these alternative execution flows.

```
[redacted]-Virtual-Machine:~/dev/ofepj$ ./run -d -b -bi B -lp 5003 -ta localhost:5002
Debug mode enabled.
Starting in mode B. Listening on port 5003
Someone connected.
Waiting for an object.
Received object of type protocols.exchange.Hello
Starting step 0 - want a Hello
Received a Hello from the user with identity A
Ended step 0.
Waiting for an object.
Received object of type javax.crypto.SealedObject
Starting step 1 - want a SealedObject.
Ended step 1.
Waiting for an object.
Received object of type java.security.SignedObject
Starting step 2 - want a SignedObject.
EOOm verification succeeded!
To resolve now type 'r'. To sign the EORm with a bogus key press 'k'. To send a bogus message press 'm'. To continue type anything else.

Ended step 2.
Waiting for an object.
Received object of type java.lang.String
Starting step 3 - want a K
Connected to the trusted third-party. Starting the resolution process.
Session resolved. The secret message is: This is my consent.
Quitting.
```

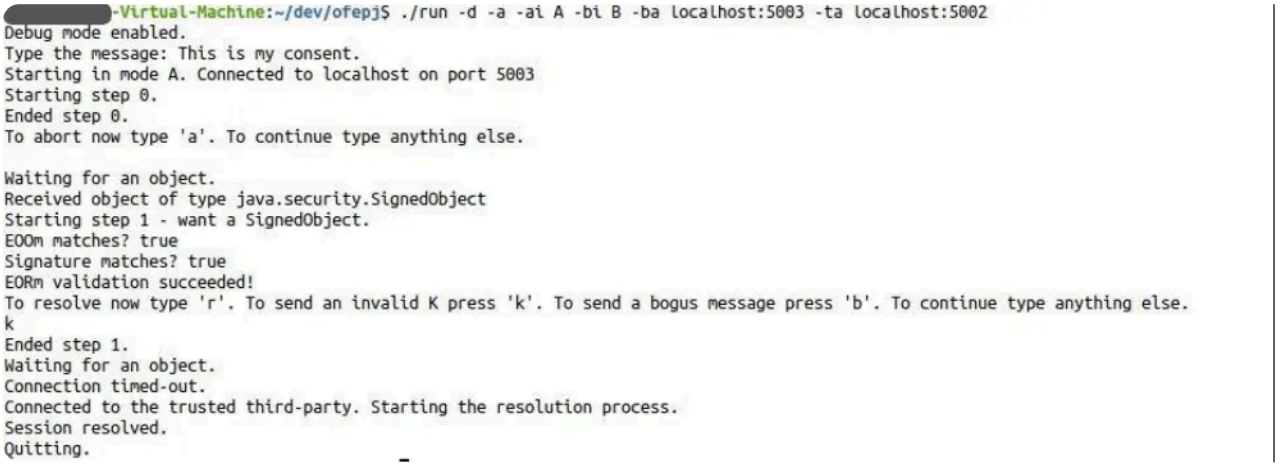
Figure 5: Protocol Execution with TTP intervention - A sends incorrect key K



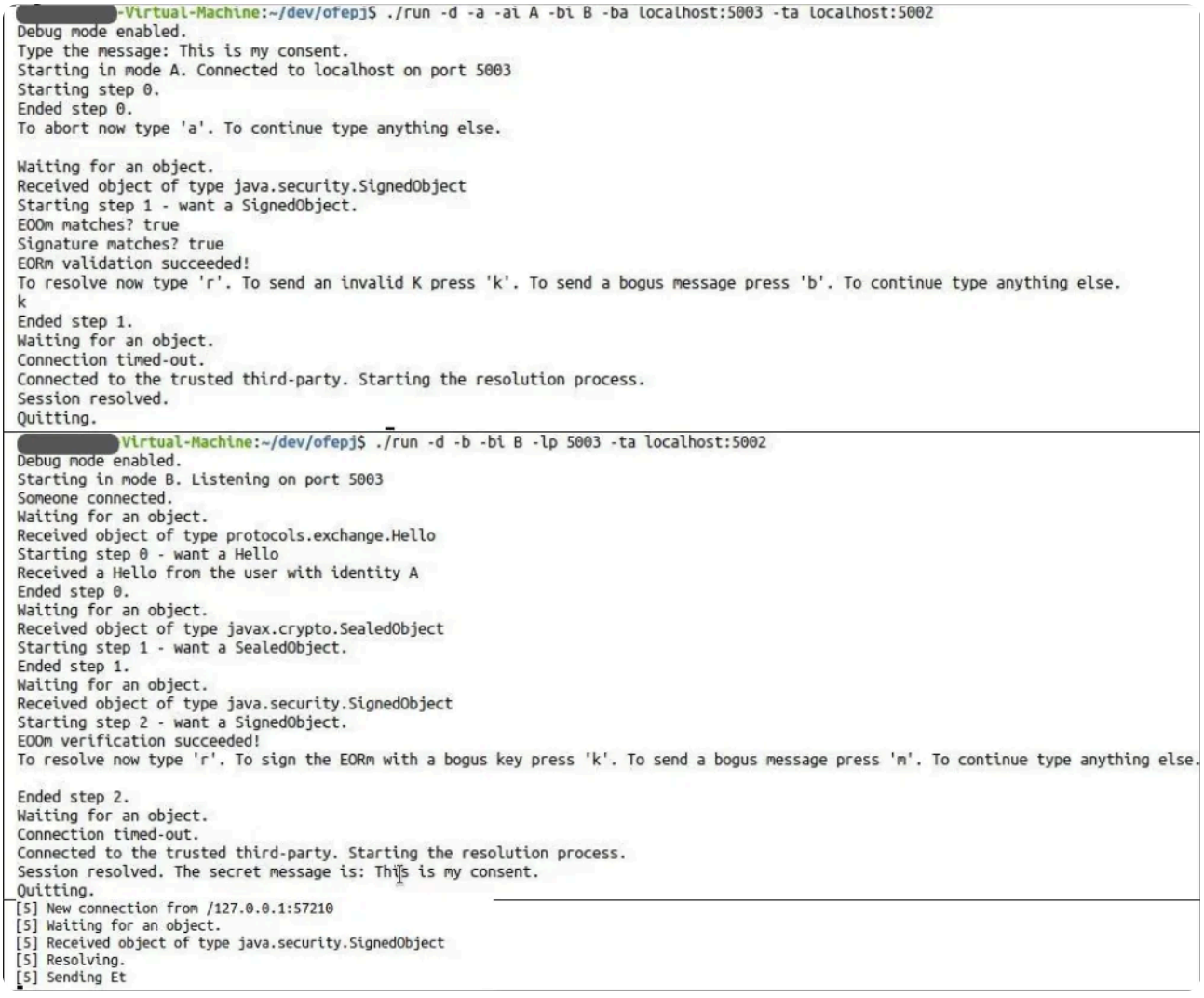
5. Conclusions

This project implemented an optimistic fair exchange protocol for data transparency logging that ensures the correctness and completeness of transparency logs. This protocol is applicable to data sharing scenarios that involve interactions with more than one party (namely consent release and data transfer). Evidences generated by this protocol can be further used by either the data controllers or data authorities to prove or check for compliance to private data regulations.

6. References







[1] H. Pagnia, "Fair Exchange," *Comput. J.*, vol. 46, no. 1, pp. 55–75, Jan. 2003, doi: 10.1093/comjnl/46.1.55.

[2] S. Even, O. Goldreich, and A. Lempel, "A Randomized Protocol for Signing Contracts," *Commun. ACM*, vol. 28, no. 6, pp. 637–647, Jun. 1985, doi: 10.1145/3812.3818.

[3] M. Ben-Or, O. Goldreich, S. Micali, and R. L. Rivest, “A Fair Protocol for Signing Contracts,” *IEEE Trans. Inf. Theory*, vol. 36, no. 1, pp. 40–46, 1990, doi: 10.1109/18.50372.

[4] S. Kremer and O. Markowitch, “Fair multi-party non-repudiation protocols,” *Int. J. Inf. Secur.*, vol. 1, no. 4, pp. 223–235, Jul. 2003, doi: 10.1007/s10207-003-0019-3.

[5] M. Abadi, N. Glew, B. Horne, and B. Pinkas, “Certified Email with a Light On-line Trusted Third Party: Design and Implementation,” in *Proceedings of the eleventh international conference on World Wide Web - WWW ’02* .

[6] B. Cox, J. D. Tygar, and M. Sirbu, “NetBill Security and Transaction Protocol.” 1995.

[7] M. O. Rabin, “Transaction protection by beacons,” *J. Comput. Syst. Sci.*, vol. 27, no. 2, pp. 256–267, Oct. 1983, doi: 10.1016/0022-0000(83)90042-9.

[8] N. Zhang, “Achieving Non-repudiation of Receipt,” *Comput. J.*, vol. 39, no. 10, pp. 844–853, Oct. 1996, doi: 10.1093/comjnl/39.10.844.

[9] A. Bahreman and J. D. Tygar, “Certified electronic mail,” in *Proceedings of the Internet Society Symposium on Network and Distributed System Security*, 1994, pp. 3–19.

[10] J. Zhou and D. Gollmann, “An efficient non-repudiation protocol,” in *Proceedings - IEEE Computer Security Foundations Symposium*, 1997, pp. 126–132, doi: 10.1109/CSFW.1997.596801.

[11] T. Coffey and P. Saidha, “Non-repudiation with mandatory proof of receipt,” *Comput. Commun. Rev.*, vol. 26, no. 1, pp. 6–17, Jan. 1996, doi: 10.1145/232335.232338.

[12] N. Asokan, M. Schunter, and M. Waidner, “Optimistic protocols for fair exchange,” in *Proceedings of the ACM Conference on Computer and Communications Security*, 1997, pp. 6–17, doi: 10.1145/266420.266426.

[13] N. Asokan, V. Shoup, and M. Waidner, “Asynchronous protocols for optimistic fair exchange,” in *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, 1998, pp. 86–99, doi: 10.1109/secpri.1998.674826.

[14] S. Micali, “Fair Electronic Exchange with Virtual Trusted Parties.”

[15] N. Asokan, V. Shoup, and M. Waidner, “Optimistic fair exchange of digital signatures,” *IEEE J. Sel. Areas Commun.*, vol. 18, no. 4, pp. 593–610, Apr. 2000, doi: 10.1109/49.839935.

[16] G. Ateniese, “Efficient verifiable encryption (and fair exchange) of digital signatures,” in *Proceedings of the ACM Conference on Computer and Communications Security*, 1999, pp. 138–146, doi: 10.1145/319709.319728.

[17] F. Bao, R. H. Deng, and W. Mao, “Efficient and practical fair exchange protocols with off-line TTP,” in *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, 1998, pp. 77–85, doi: 10.1109/secpri.1998.674825.

[18] M. Franklin and G. Tsudik, “Secure group barter: Multi-party fair exchange with semi-trusted neutral parties,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1998, vol. 1465, pp. 90–102, doi: 10.1007/BFb0055475.

[19] Feng Bao, R. Deng, K. Q. Nguyen, and V. Varadharajan, “Multi-party fair exchange with an off-line trusted neutral party,” Aug. 2008, pp. 858–862, doi: 10.1109/dexa.1999.795294.

[20] N. Asokan, N. Asokan, M. Schunter, and M. Waidner, “Optimistic Protocols for Multi-Party Fair Exchange,” 1996, Accessed: Jan. 08, 2021. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.1415.

[21] O. Markowitch and S. Kremer, “A multi-party optimistic non-repudiation protocol,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2001, vol. 2015, pp. 109–122, doi: 10.1007/3-540-45247-8_9.

[22] C. C. Oniz, E. Savas, and A. Levi, “An optimistic fair E-commerce protocol for large Egoods,” in *Proceedings of ISCN’06: 7th International Symosium on Computer Networks*, 2006, vol. 2006, pp. 214–219, doi: 10.1109/ISCN.2006.1662536.