Inter-FSP Funds Transfer Protocol

Technical Report

Amir Herzberg and Shay Nachmani

Computer Science Department, Bar Ilan University

Table of Contents

Abstract. We present the first *decentralized secure funds transfer protocol* with multiple participants. The protocol ensures that a participant can only lose money, if a peer it trusted is corrupted. Furthermore, the loss is bounded by the credit allocated to this partner. The protocol supports expiration times for payment orders, and realistic network queuing delay. We achieve this using model and techniques from the Quality of Service area to guarantee delays and avoid payment order expiration. We present rigorous security proof.

1 Introduction

In this work we focus on electronic funds transfer, which is a basic and critical economic operation. We use the term *funds transfer* when speaking on money that transfers between FSPs [1], rather than *payments* that usually come against some goods.

Consider a client that wish to buy some goods from vendor A in the Internet. The client cannot pay A directly, since there is no payment mechanism between them. So, the client must deliver some commitment to payment from FSP B that A works with. This funds transfer operation consists of several stages, which are illustrated in Fig. 1. First, FSP B issues Payment Certificate (PC) for its partner A. PC is an electronic signed certification that B commits to transfer funds against Payment Orders (PO), under some restrictions. PO is an electronic signed message that orders a payment of money. The restrictions in the PC may include limiting the total amount of the PO's, a deadline of the commitment, a commission rate, etc. Also, the PO itself may include expiration date. B may issue such PC upon a request from its client, that deposited money in B and wants to transact with A. The client can pay to A with PO that B would issue for him. Then, when A gets this PO, it can be redeemed to B, which should transfer the funds as it was committed.

The scope of our protocol is restricted to the stages of issuing the PC, and redeeming the PO, these stages are illustrated in Fig. 1 with solid lines. The physical payment and goods transfer between peers are out of band processes, illustrated with dashed lines in Fig. 1. Though, our protocol is informed about these operations, and reacts upon this information.

There is some complexity with this operation. First, *trust* is needed between two parties that wish to transact with each other. A party cannot ensure that its peer will pay the money as it committed, whether because the peer is corrupted or bankrupted.

Second, payment commitments require reserving funds to cover them, this implies the need for expiration time of the commitment, so that reserved funds can be freed. As a result, there is a communication dependence, if a party fails to send redemption, or if the network is overloaded and the redemption expires

[1] Finance Service Providers are financial institutes, e.g. banks, insurance companies and investment funds, that have business relationships with each other and handling accounts for their partners.

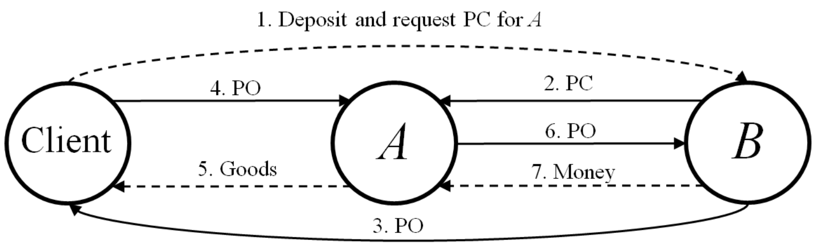

Fig. 1. Electronic funds transfer operation, where B commits with PC to pay money to A against PO's, and the client pays to A with B's PO

before it is received, it might cause some party to lose money. So, the end-to-end communication delay should be taken into consideration. Of course, queuing delay depends on the load of the network, and thus, to bound it we need to use QoS techniques.

These problems can be dealt rather easily, if all parties have trust relationships with some Trusted Third Party (TTP), which is trusted to pay or to resolve communication conflicts between the parties. Most of the current systems and research solutions take this centralized approach. Centralized designs are simple to design, launch and operate; however, they also have significant disadvantages, motivating us to focus in this work on decentralized design, we list below several disadvantages:

Lack of competition Clearly, the TTP takes commission from the parties that wish to transfer funds with each other. Naturally, there are very few TTP that both sides trust on. This situation yields lack of competition, and thus, relatively high costs.

Reduced interoperability In the centralized model there may be entities that do not have common TTP, and thus, they cannot perform any economic operation with each other.

Bottleneck Centralization creates significant load and overhead on the TTP, since all the traffic goes through it.

Single point of failure (and 'legal attack') A failure in the TTP might cause disruption of the whole network, which yields Denial of Service vulnerability. An illustrating example is the European Council instruction to SWIFT to stop providing services to Iranian banks subject to European Union sanctions [1], that should make trading with Iran impossible.

We present the first provably secure decentralized funds transfer protocol. Our solution allows transactions through multiple FSPs over the Internet, where each FSP interacts and trusts only its partners, without relying on one common trusted third party. The decentralization should increase the competition between payment providers, reduce fees and costs, and improve availability and connectivity.

Decentralization also makes it more likely for smaller players to participate. Therefore, the assumption that trading systems are all large, highly reputable institutes, iss no longer justified. The smaller parties may misbehave, and the decentralized model creates new challenges, where several parties might cooperate against other party.

1.1 The Network Limitation Aspects

The topic of Quality of Service in networks has been researched widely, however, the influence of network delays on electronic trade has not been explored yet.

Under some assumptions on the network, e.g. minimal rate, we ensure that if the party behaves according to the proposed protocol, it will never lose money due to excessive network delays.

In order to understand the communication challenges in decentralized system, consider the case illustrated in Fig 2, with four FSPs: *A*, *A'*, *B*, and *C*. *B* receives PO's from *A* and *A'*, and redeems them to *C*. Who will lose the money if *A* and *A'* would redeem PO's to *B*, in greater rate than *B* can handle? Also, if the rate that *B* receives the PO's is greater than the rate it can redeem to *C*, what should happen if *A* or *A'* redeem hundreds of PO's just before the deadline of them expires, and B fails to redeem them to *C*? Furthermore, if *A* would redeem to *B* a PO just before it expires, *B* won't succeed to send it to *C* before the deadline. If we decide that *B* should lose, the other parties *A*, *A'* and *C* can misbehave and cooperate to make *B* lose. Nevertheless, if we decide that *A* or *A'* should lose, *B* and *C* can misbehave and cooperate to make *A* or *A'* lose.

We note that when the participates (e.g. *A*,*A'*,*B*,*C*) are reputable entities e.g. banks, clearly, they would not abuse the system like that. However, we design the protocol to be secure even for less trust worth participates(smaller, less reputable etc.). We can avoid these losses using risk limiting with QoS techniques.

Our solution, for the first time, addresses the queuing and congestion delays, namely considers that the network is capacity limited. This issue is vital when speaking on delivering payment orders and commitments with expiration dates. We assume that the underlying layer guarantees delivering messages in a minimal rate with some latency. This popular model is called 'Latency Rate Server' [12], and it will be discussed in the communication model (Section 3.3).

Based on the Latency Rate Server, the protocol limits the rate of redemptions. That is, one of the restrictions of the PC is the maximal rate of using it. The formulation of this rate restriction is given by the 'Leaky Bucket Model',[12] (Section 4.2), which allows to set maximal rate and maximal pending request bucket, that limits the transmissions. This way, an honest peer can limit its PC, according the rate restriction of the next peer in the redemption path.

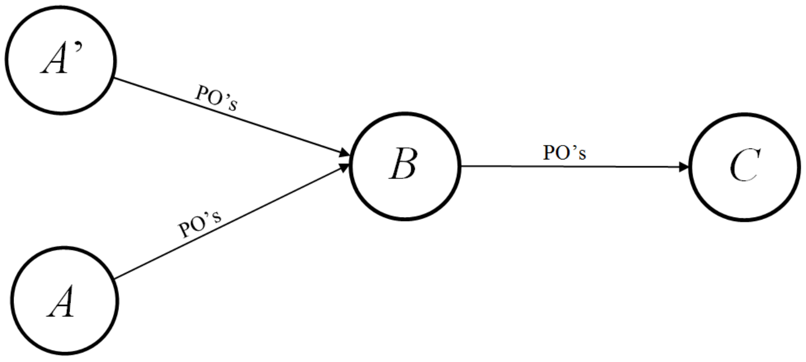

Fig. 2. Funds Transfer with Four FSPs

By limiting transmissions, using the Leaky Bucket model, the protocol limits the maximal delay. Hence, a peer can calculate the extra treatment time that is needed for redeeming PO before it expires.

1.2 The Trust Model

When financial institutes such as banks do business with each other, they decide what is the credit for each partner. This credit expresses the maximum amount that would be risked, trusting on this partner, taking into account the reputation, and the probability for bankruptcy of the peer or fraud by the peer or its employees. The proposed protocol provides automated risk limiting, where the risks are externally determined by the upper layer. Each party in the protocol sets credit for each peer, and the protocol ensures that the loss caused by the peer is limited by this credit. Moreover, the protocol guarantees that a party can lose money only if its peer is corrupted, namely risk limiting is *conservative*, i.e. we require from honest participate to limit the total risk for all its peers to be lower than amount that may cause bankruptcy. Of course, if all the parties performs the protocol correctly, there will never be a loss.

In order to understand the trust model, consider the following scenario that illustrated in Fig 3. FSP A gives FSP B credit of 1M\$, namely A allows B to owe it 1M\$. B can use this credit to issue PC for his clients and commit to pay A against its PO's. A will respect these PO's up to 1M\$ until B pays it the money.

Now consider that B gives credit of 2M\$ to FSP C, and C issues PC for B. In this situation, B can issue PC to A with a commitment to redeem also PO's from C up to 2M\$. This way, funds can transfer from C to A, B pays against C PO's to A, and get its payment from C. Of course, the PC's have deadlines, and limited amounts. Therefore, the protocol must limit the commitment B can give in PC, based on C's PC, to avoid loss. Also, as we mentioned above, there

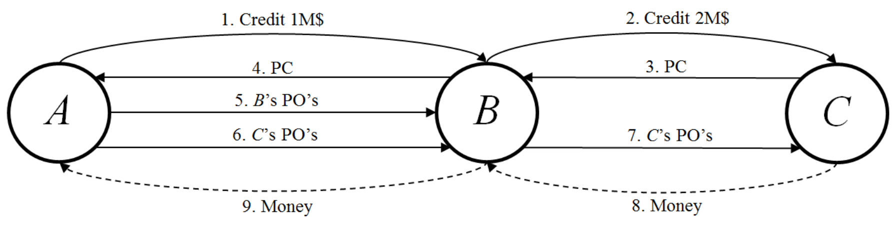

Fig. 3. Protocol Scenario with Three FSPs

are network limitations, and B should limit the rate of PO's redeemed from A, so these PO's could be redeemed by C successfully. Additionally, if B would like to issue PC to another peer A' with a commitment to redeem C's PO's (like in Fig. 2), it should divide the credit for C between A and A' in its PC, and to divide the redemption rate and the amount from C's PC as well.

Obviously, fraud is a prevalent concern with online transactions, double spending, forgery, and repudiation must be avoided. Therefore, we will present rigorous cryptographic proof of the security of the payment system. Also, we provide general model for security requirement validation of a protocol. Although a variety of automatic proof systems, we did not find a system that can model the service rate limitation, and thus to prove the network delay requirement.

1.3 Our Contributions

We present decentralized funds transfer protocol, allowing multiple FSPs.

- The protocol guarantees that for each party a loss can be caused only by corrupted peer, and that is limited by the given credit for this peer.
- We address the network dependency aspect, taking into account the communicative capabilities limitations of the participates, and the communication delay effect on the financial commitments.
- We use a well defined generic model for requirements validation, that take into account the assumed communication model of the lower layer. This model can be used for other protocols.
- The protocol supports fee charge by the FSP over the redemption path, for the service of transferring the funds.

Organization In Section 2 we review the related work. In Section 3 we describe the model of the system, the assumptions of the upper and the lower layers and their services. In Section 4 we define the requirements of the protocol. In Section 5 we present our protocol. In Section 7 we prove that the protocol meets the requirements. In section 8 we prove the liveness property of the protocol.

2 Related Works

We use communication models that are described broadly in the literature, e.g. in [12].

2.1 Funds Transfer Protocols

This work is based on the Herzberg et al. patent [9], which defines the framework of the funds transfer as we show it, but there is no security analysis there neither no network delay aspects.

There is some research works on the inter bank funds transfer, in [13] Leinonenet al. developed a payment and settlement system simulator that can be used for constructing simulation models of payment systems. In [20] Zho examined the operation flow of the electronic funds transfer process and its security control mechanism. None of these works handles the risk limiting and the network issues we deal in our work.

2.2 Electronic Payment Systems

A lot of work was made on the area of anonymity [5, 6, 7] we are not dealing with this issues at this stage, although it might be integrated in a future work.

We review below the research on open decentralized payment systems. Schmees [16] has described the benefits of leaving a centralized client- server payment system and moving toward distributed electronic payment using Peer-to-Peer networks. However, he did not propose any mechanism of this model.

Yang and Garcia-Molina describe a micropayment system protocol called PPay [19] that is built upon a Peer-to-Peer network. They present the concept of floating, self-managed currency which allows digital currency to float from one node to another without the involvement of a centralized broker. Although PPay security is based on digital signatures, fraud cannot be prevented, but mechanism of punishment and risk management is proposed for making fraud unprofitable.

Xiong et. al developed PeerTrust [18] a Peer-to-Peer trust model for quantifying and comparing the trustworthiness of peers based on a transaction-based feedback system. They build general trust metric that provides an effective measure for capturing the trustworthiness of peers, addresses the fake or misleading feedbacks, etc. minimizing the risk involved in e-commerce transactions. However, it is not providing any protection towards a certain outcome of a given transaction.

Bitcoin [14] is an electronic coin defined as a chain of digital signatures. Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin. A payee can verify the signatures to verify the chain of ownership. To to prevent double-spending, a peer-to-peer network using proof-of-work to record a public history of transactions is proposed. Fraud becomes computationally impractical for an attacker if honest nodes control a majority of CPU power.

One can easily see that none of the above works contains signed commitments between peers, with deadlines. Thus, the challenges are extremely different from our work, and in particular, none of them deals with the network delay influence on the payments.

2.3 Formal Security Proofs

There are two approaches to conduct a security proof:

The cryptographic approach Which is usually proved by reduction to one of the underlying cryptographic primitives. Although the rigor proofs this approach allows, they have to be done by hand.

The automated formal-methods approach Based on the Dolev-Yao model [8] which represents cryptography as term algebras, simplifies security proofs for large protocols. In [3] Backes proved that the real cryptographic library ,with its much more comprehensive adversary, is as secure as the ideal cryptographic library. So, protocol proved on the basis of the deterministic, Dolev-Yao-like ideal library can be safely implemented with the real cryptographic library. For example, in [2] Backes used this model for proving 3KP electronic payment system [4]. We used these results for the security analysis of our protocol. Also, based on the work of Herzberg and Yoffe [11], we defined a security requirement framework, for our protocol security analysis.

3 General Security Model

In this section we provide the model of the execution of the protocol, and present the general model of validation of any set of requirements under any set of communication assumptions. This model can be used for various protocols and other missions with different requirements and different communication models. This model is based on the work of Herzberg and Yoffe [11].

3.1 Protocol Execution Model

Each party is represented by interacting machine $\mathcal{M}$ which react on received input [15].

A machine is a probabilistic I/O automaton (extended finite-state machine) in a slightly refined model to allow complexity considerations. For these machines Turing-machine realizations are defined, and the complexity of those is measured in terms of a common security parameter 1^k, given as the initial work-tape content of every machine. We only use polynomial run-time deterministic I/O automata, which its output is also polynomial by the initial input. This brief definition is taken from [3]. The machine is controlled by its administrator, by internal input commands. Also, the machine raises relevant information to the administrator, upon received external input.

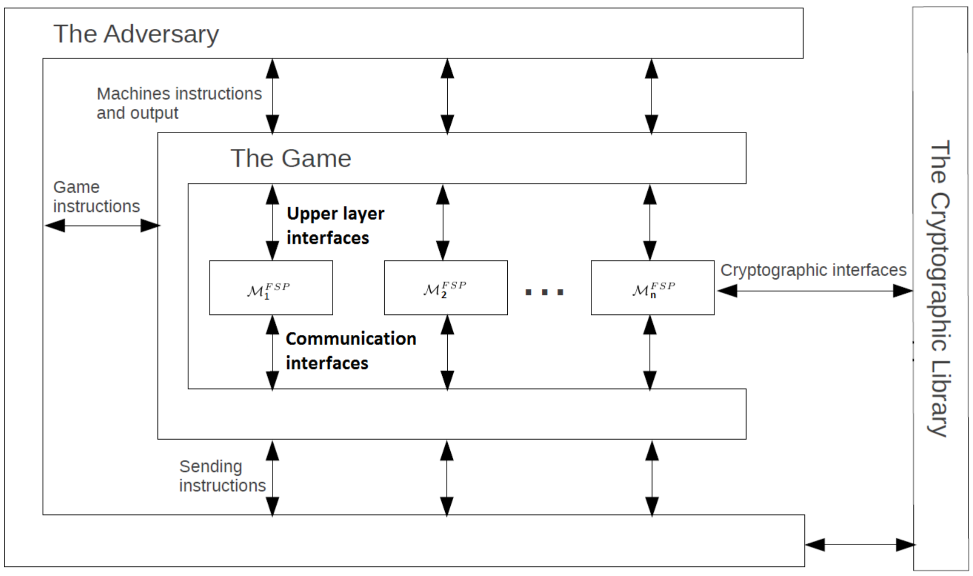

Fig. 4. Overview of the Game

3.2 Requirements Model

We define a game in which the adversary sets honest parties, and corrupted parties. The honest parties are following the protocol, with the adversary as administrator, also, they have reliable communication with each other. The corrupted parties are fully controlled by the adversary. The honest parties and the adversary are using the cryptographic library directly. The adversary controls all the communication at the game. The time is global for all machines, it is received from the adversary, and it verified to be non-decreasing. See Fig 4. for the game overview.

The game gets the adversary algorithm as input, and executes its instructions step by step. These instructions might be sending messages to honest parties, or giving administrator instructions for them. The adversary is is a machine $\mathcal{A}$ that has the following interfaces:

- $(\texttt{init\_request}, 1^k)$ gets the security parameter 1^k and returns the tuple $(state, communication\_parameters, n)$ where $state$ is the state of the adversary machine. $communication\_parameters$ are the parameters of the communication model, and n is the number of honest machines $\mathcal{M}$ to initialize by the game.
- $(\texttt{next\_state}, adv\_state)$ gets the current $\mathcal{A}$ state and returns the tuple $(instruction, id, parameter\_list, time, adv\_state)$ where $instruction$ may be a name of instruction to $\mathcal{M}$, or instruction to the game: to finish the game

or to send a message to some party. *id* is the identifier of the $\mathcal{M}$ that should get this instruction or message. *parameter_list* is a list of the instruction parameters. *time* is the current time, and *adv_state* is the next $\mathcal{A}$ state.
- (`protocol_output`, *out*, *sending_request*, *adv_state*) gets the $\mathcal{M}$ output and sending request with the current state, and returns the next state of $\mathcal{A}$.

The game ensures that the time is strictly increasing, and more than one operation cannot take place at the same time.

The game provides cryptogaphic library to the parties and the adversary. This library gets the security parameter, and includes interfaces for generating randomize id and key pairs, as well as interfaces for signing and verifying messages which can be broken with negligible probability with respect to the security parameter. An example for such library can be found in [3].

The game saves log for the communication and log for the protocol, where every instruction for honest party and every output of it is logged. When the adversary gives instruction to finish the game, the game returns True if the adversary lost, i.e., the protocol log is valid (no cheating), or the communication log is invalid (adversary did not preserve the assumptions about the communication layer).

More precisely, given a log of running $\mathcal{M}$ over n honest parties, we require that

$$Valid\_ProtocolLog(ProtocolLog) = True$$

as long as

$$Valid\_CommLog(CommunicationLog) = True$$

These functions should be implemented specifically for some protocol and communication model. The game gets these functions as parameters. We give our implementations in the next section. For the game pseudo code, see Algorithm 1 below.

Definition 1 *We say that machine $\mathcal{M}$ is a* secure machine *with respect to $Valid\_CommLog$ and $Valid\_ProtocolLog$, if for every polynomial adversary $\mathcal{A}$, the probability*

$$Pr[Game(\mathcal{M}, Valid\_CommLog, Valid\_ProtocolLog, \mathcal{A}, 1^k) = False]$$

is negligible with respect to the security parameter 1^k.

3.3 The Communication Model

We assume simple model, in which the machines communicate over network that its peers can serve requests from each other with some delay and afterwards with a fixed rate of service. This model is called Latancy Rate Server [17] which is a general model for the analysis of scheduling algorithms. It enables calculating tight bounds on the end-to-end delay of individual sessions in an arbitrary network of schedulers. LR server model is widely used in the literature and many

Algorithm 1: Game(machine $\mathcal{M}$,$Valid\_CommLog$,$Valid\_ProtocolLog$, adversary $\mathcal{A}$, security_parameter 1^k)

```
1  $ProtocolLog = ""$;
2  $CommLog = ""$;
3  $CurrentTime = 0$;
4  $H = \phi$;
5  $(adv\_state, communication\_parameters, n) = \mathcal{A}(\text{init\_request}, 1^k)$;
6  for $i = 1 \to n$ do
7      $(state, id) = \mathcal{M}^{FSP}(\text{init\_request}, 1^k, communication\_parameters)$;
8      $States[id] = state$;
9      $H = H \cup \{id\}$ ;
   // The first line of the log is for the parameters
10 $ProtocolLog+ = (\text{Parameters}, communication\_parameters, n, H)$;
11 $CommLog+ = (\text{Parameters}, communication\_parameters)$;
12 while True do
13     $(instruction, id, parameter\_list, time, adv\_state) =$
       $\mathcal{A}(\text{next\_state}, adv\_state)$;
14     if $time \leq CurrentTime$ then
15         return False;
16     $CurrentTime = time$;
17     if $instruction = \text{finish\_game}$ then
18         $ProtocolLog+ = (\text{finish\_game}, time)$;
19         return Valid_CommLog($CommLog$) $\wedge$!Valid_ProtocolLog($ProtocolLog$)
20     if $instruction = \text{send}$ then
21         $(u, msg) = parameters\_list$;
           // sending_request is a (destination, message) list
22         $(States[id], sending\_request, out) = \mathcal{M}^{FSP}(\text{received}, States[id], u, msg)$;
23         $CommLog+ = (\text{received}, time, id, u, msg)$;
24     else
           // regular instruction to machine
25         $(States[id], sending\_request, out) =$
           $\mathcal{M}^{FSP}(instruction, States[id], parameters\_list)$;
26         $ProtocolLog+ = (id, time, instruction, parameters\_list)$;
27     $adv\_state = \mathcal{A}(\text{protocol\_output}, out, sending\_request, adv\_state)$;
28     foreach $(dest, msg) \in sending\_request$ do
29         $CommLog+ = (\text{send}, time, id, dest, msg)$;
30     $ProtocolLog+ = (id, time, out)$;
```

Algorithm 2: LR_FIFO_Valid_CommLog($CommLog$)

```
1  set $L, R$ to the values in (Parameters, $L, R$) ;
   // CommLog next lines are of the form (send, source,
      destination,time,message) or
      (received,destination,source,time,message )
   // We require LR server with the input parameters
2  Let $R_{u,v,t_1,t_2} = |\{(\texttt{received}, u, v, t, msg) \in CommLog | t_1 \leq t \leq t_2\}|$
3  Let $S_{u,v,t_1,t_2} = |\{(\texttt{send}, u, v, t, msg) \in CommLog | t_1 \leq t \leq t_2\}|$
4  if $\exists (u, v, t_1, t_2) | \exists t_1 \leq s \leq t_2 \land R_{v,u,t_1,t_2} < S_{u,v,t_1,s} - (t_2 - s - L) * R$ then
5      return False;
   // We require FIFO
6  Let $R\_MSG_{u,v,t_1}$ be the concatenation in chronological order of all the
   messages in $\{msg | (\texttt{received}, u, v, t, msg) \in CommLog \land t \leq t_1\}$
7  Let $S\_MSG_{u,v,t_1}$ be the concatenation in chronological order of all the
   messages in $\{msg | (\texttt{send}, u, v, t, msg) \in CommLog \land t \leq t_1\}$
8  if $\exists (u, v, t) | R\_MSG_{v,u,t}$ is not prefix of $S\_MSG_{u,v,t}$ then
9      return False;
10 return True;
```

well-known scheduling algorithms, such as TDM, Fair Queueing, VirtualClock, Weighted Round Robin ,etc. have been proved to belong to the class of LR-servers. The behavior of Latancy Rate Server is determined by two parameters L and R as follows: After initial latency L, the scheduler allocates service rate of minimum R to the session. More precisely, the model guarantee delivering $(t - L) * R$ bits in interval of length t. For graphic description see Fig. 5. We assume that all the links are with the same parameters, to avoid needless complexity. Also, we assume reliable FIFO communication between parties in the protocol. We give our implementation to the $Valid\_CommLog$ function of the game in Algorithm 2 and we call it $LR\_FIFO\_Valid\_CommLog$. This function validates the follows:

1. Communication service rate according to LR server model with the input parameters. (lines 2 - 5)
2. Reliable FIFO communication (lines 6 - 9).

4 FSP Machine Requirements

In this section we define the FSP machine and the requirements of the protocol by implementing the generic function of the game.

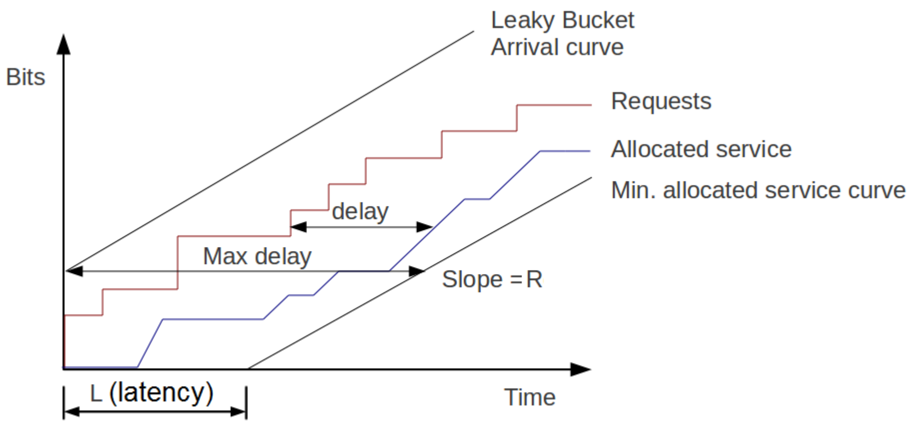

Fig. 5. Latency Rate Server

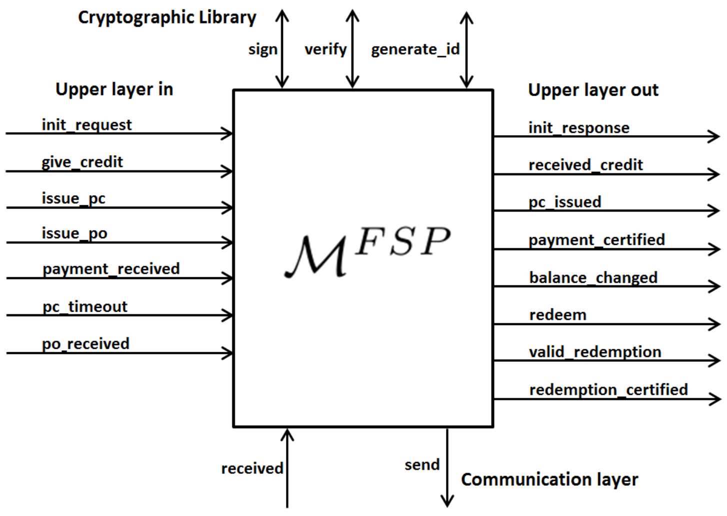

Fig. 6. $\mathcal{M}^{FSP}$ interfaces

4.1 The FSP Machine

The FSP machine is used for managing a FSP entity, it saves the balance for each peer, handles the commitments and communicate with the other peers.

The machines have interface for knowing each other by public keys inserted by the administrator. Also, they have interfaces for giving credit, sending commitments, redeeming, and certifying external payments.

The underlying layer, gives us interfaces for sending and receiving messages. The cryptographic library gives interfaces for signing and verifying signatures, and for generating identifiers and keys. We define the protocol by the FSP machine.

There is a network of FSPs which give credit for each other, that is, FSP_u administrator gives his machine command to set credit for FSP_v. Clearly, FSP_u machine must know FSP_v public key and address, that should be received in the same interface as identifier. The meaning is FSP_u trusts FSP_v and agrees that FSP_v will owe him money, no more than the given credit, i.e. FSP_u takes the risk that FSP_v will not pay as committed due to bankruptcy, corruption, or other failure.

In case the FSP machine has not get any input of credit for some partner, the credit for him is zero. The machines issue signed commitments - PCs, and checks - POs, and redeem them. Also, the upper layer can notice the machine on external payment, and certification about this payment should be sent to the payer machine. For readability, we call the inputs from the upper layer interfaces - instructions.

Definition 2 $\mathcal{M}^{FSP}$ *is a machine with the following interfaces:*

1. Upper layer interfaces
   - $in(\texttt{init\_request}, security\_parameter, communication\_parameters)$ for initializing FSP machine with the communication parameters.
   - $out(\texttt{init\_response}, identifier)$ for delivering the identifier given by the underlying layer to the upper layer.
   - $in(\texttt{give\_credit}, peer\_identifier, credit, b, r)$ for giving initial credit, and setting Leaky Bucket parameters b, r to limit the communication rate to this peer. The Leaky Bucket model is discussed below in Section 4.2.
   - $out(\texttt{received\_credit}, peer\_identifier, credit)$ for reporting of given credit.
   - $in(\texttt{issue\_pc}, PC\_arguments)$ for issuing signed commitment - PC (Definition 5).
   - $in(\texttt{pc\_timeout}, PC_{id})$ for notifying of PC expiration.
   - $out(\texttt{pc\_issued}, payment\_certificate)$ for reporting of received PC (Definition 5).
   - $in(\texttt{payment\_received}, peer\_identifier, amt)$ external payment receiving notification for the payee.
   - $out(\texttt{payment\_certified}, peer\_identifier, redemption\_id, amt)$ for payer to report on receiving external payment certification from the payee.
   - $out(\texttt{redeem}, peer\_identifier, redemption\_id, amt)$ informs the upper layer on sending redeem request of signed check - PO (Definition 7).

- out(`valid_redemption`, $peer\_identifier, redemption\_id, amt$) for reporting on a valid redemption request that came from other peer.
- out(`redemption_certified`, $peer\_identifier, redemption\_id, amt$) for reporting on redemption certification that came from the peer that the redeem request was sent to.
- in(`issue_po`, $path, amt, expire$) for issuing PO.
- in(`po_received`, PO, PC_{id}) for redeeming PO that came from upper layer against PC.

2. Communication interfaces
   - out(`send`, $peer, message$)
   - in(`received`, $peer, message$)
3. Cryptographic interfaces
   - (`generate_id`, $security\_parameter$) generates unique identifier and key pair for signing.
   - (`sign`, $key, message$) creates an electronic signature of a message.
   - (`verify`, $message, key, signature$) verifies an electronic signature of a message with key, returns true upon successful verification, and false otherwise.

The interfaces are illustrated in Fig. 6.

4.2 The Leaky Bucket model

We use the Leaky Bucket model to limit arrival rate between peers. The model gets rate r, and bucket size b as parameters, and commit to service packets if the requests arrival rate is less then r, with buffer size of b. A validation pseudo code of Leaky Bucket for limiting requests rate in the machines is provided in Algorithm 3.

Algorithm 3: LB(b,r)

```
Initially:
    $bucket = 0$ ;
Upon request arrival:
    if $bucket + 1 \leq b$ then
        $bucket + +$;
        return True;
    else
        return False;
Every 1/r seconds:
    if $bucket > 0$ then
        $bucket - -$;
```

Together with the LR server model, we can bound the maximal delay between peers, where the communication is limited by LB model parameters.

Definition 3 *Let $Delay(u, v)$ denote the upper bound of the communication delay from u to v, computed based on the LR server model parameters, L, R, and the LB parameters that come from the instruction* $(\texttt{give\_credit}, v, credit, b_{u,v}, r_{u,v})$ *in u machine.*

$$Delay(u, v) = L + b_{u,v}/R$$

This would hold, only if $r_{u,v} \leq R$.

4.3 FSP requirements

We require that honest party will lose money only if its peer is corrupted, and the loss is bounded by the given credit for this peer.

Definition 4 *We define* money loss *as the scenario in which a party sends redemption request to its peer, and that peer does not accept it.*

Thus, the protocol must ensure that honest party always accept valid redemption. Also, to bound the loss, we require limitation on the possible debt, which is the negative balance, for each peer. This limitation is set by the upper layer, as the credit for that peer.

To summarize, the requirements are the following:

1. Every valid redemption to an honest peer must be accepted.
2. An honest party will never have negative balance for a peer, that breaks the credit limits for this peer.

To validate these requirements we implement the $FSP\_Valid\_ProtocolLog$ function of the game in Algorithm 4.

First, lines 2 - 3 set there the initial credit, balance and delay to zero for each peer.

Then we go over the *ProtocolLog*, line by line (line 4).

Lines 5- 10 set the credit values that each honest peer gives to its partners, and the calculate the maximal communication delay from u to v.

Line 19 in Algorithm 4 validates the first requirement, we check that every redeem request by honest party to an honest peer is accepted, no later than the maximal communication delay. Of course, if the game was finished before the redeem message arrived, we do not require anything.

For the validation of the second requirement, we update the values of the balance of the parties, upon redemptions (lines 16, 18) and payments (lines 24, 26). The condition at line 27 validates that negative balance does not break the credit limits as required.

The condition at line 22 are for restricting the external payment notifications to zero negative balance, i.e. the payer cannot gain positive balance with external payment. This way, the adversary cannot make negative balance of the payee, what might break the credit bounds.

Algorithm 4: FSP_Valid_ProtocolLog($ProtocolLog$)

```
1  set $L, R, n, H$ to the values in (Parameters, $L, R, n, H$) $\in ProtocolLog$ ;
   // ProtocolLog next lines are of the form (FSP identifier,time,
      instruction, parameter list)
2  $Credit[n][] = 0$;
3  $Delay[n][] = 0$ ;
4  foreach $line(u, t, line\_type, v, paramters) \in ProtocolLog$ do
       // we go over ProtocolLog lines sequentially, u must be honest
5      if $line = (u, t,$ give_credit, $v, k, credit, b, r)$ then
           // We allow only one give_credit instruction for each party
6          if $Credit[u][v]! = 0$ then
7              return True;
           // We allow only non-negative credit
8          if $credit < 0$ then
9              return True;
10         $Credit[u][v] = credit$;
           // We allow LB rate up to the communication rate
11         if $(b < 0) \vee (r < 0) \vee (r < R)$ then
12             return True;
           // max delay from u to v
13         $Delay[u][v] = L + b/R$ ;
14         $Balance[u][v] = 0$ ;
15     if $line = (u, t,$ valid_redemption, $v, id, amt)$ then
16         $Balance[u][v]+ = amt$ ;
17     if $line = (u, t,$ redeem, $v, id, amt)$ then
18         $Balance[u][v]- = amt$ ;
           // Every valid redemption to an honest peer must be validated.
19         if $(v \in H) \wedge\ (!\exists t_2 \leq t + Delay(u, v))\ |$
           $(\exists\,(v, t_2,$ valid_redemption, $u, id, amt) \in ProtocolLog \vee \exists\,($finish_game, $t_2) \in ProtocolLog)$
           then
20             return False;
21     if $line = (u, t,$ payment_received, $v, amt)$ then
22         if $Balance[u][v] + amt > 0$ then
23             return True;
24         $Balance[u][v]+ = amt$ ;
25     if $line = (u, t,$ payment_certified, $v, amt)$ then
26         $Balance[u][v]- = amt$ ;
       // Credit bounds the negative balance
27     if $Credit[u][v] < -Balance[u][v]$ then
28         return False;
29 return True;
```

5 The Inter-FSP Funds Transfer Protocol

In this section we give an overview to the Inter-FSP Funds Transfer Protocol which is our implementation of $\mathcal{M}^{FSP}$ machine, as illustrated in Fig. 7.

In Section 6 we formalize it with detailed description and in Section 6.4 we present accurate and efficient algorithms of the protocol implementation.

An FSP is identified by the tuple $< addr, pub >$ which is given by the underlying cryptographic library, where *addr* is the address for the communication layer, and *pub* is the public key which is used to sign and verify the FSP's messages.

5.1 Give Credit

The first stage of the FSPs protocol is giving credit to each other. As explained previously, the credit that FSP A gives to its peer B is the amount that A risks trusting on B, i.e. it bounds the loss A would have if B misbehaves. Setting the credit for some peer $< f, k >$ is done by the administrator instruction $(\texttt{give\_credit}, f, k, c, r, b)$, where c is the given credit, f is the address of the peer and k is its public key. By this instruction, the upper layer also sets the Leaky Bucket parameters, b, r, for limiting the communication rate of sending to this peer.

The implementation is in Algorithms 8, 10 .

5.2 PC and PO

The next stage in the protocol is issuing Payment Certificates (PC). Generally, PC is a commitment to pay money against specific Payments Orders (PO) under some conditions.

Definition 5 *We define PC as the tuple:*

$$PC =< id, by, for, path, expire, trt, max, net, b, r, v, sig_{by} >$$

The meaning of the attributes are as follows:

- id is the identifier of the PC at the issuer machine.
- by is the identifier of the FSP that issued this PC.
- for is the identifier of the FSP that this PC was issued for.
- $path$ is a list of FSPs identifiers, through this path a payment should be passed.
- max is the maximal amount that should be payed using this PC.
- $expire$ is the expiration date of this PC.
- trt is the treatment time of this PC, .
- net is a fee function of using this PC with some payment amount, i.e the net amount that will be paid against redemption of amount x is $net(x)$.
- b and r are parameters of the Leaky Bucket that limits the rate of using this PC by the peer by for redemptions.
- v is a key for validation of signed Payment Order.
- sig_{by} is a signature of the issuer on the PC.

PC flavors

Definition 6 *We distinguish between two flavors of PCs:*

1. Prime *PC - This is a PC that the upper layer instructed to issue independently, with its terms and conditions.*
2. Derived *PC - This is a PC that the upper layer instructed to issue, based on another PC that was received from other peer. In addition, we call the PC that the derived PC is based on -* Base *PC.*

Definition 7 *We define Payment Order as the tuple:*

$$PO =< id, issuer, path, amt, expire, sig_{issuer} >$$

With the following attributes:

- *id* is the identifier of the *PO* at the issuer machine.
- *issuer* is the identifier of the FSP that issued this *PO*.
- *path* is a list of FSPs identifiers, through this path a payment should be passed.
- *amt* is the amount of the payment.
- *expire* is the expiration date of this *PO*.
- sig_{issuer} is a signature of the issuer on the *PC*.

Actually, the *PC* is the signed commitment that contains the terms of accepting *PO*'s. When party accept redemption of *PO* it updates the balance for the redeeming peer, and send it a certificate the the redemption was accepted.

Now we specify the conditions of *PO* to be accepted with respect to *PC*.

Let $ACCEPTED(PC)^t$ denote the set of the *PO*s that were accepted with respect to *PC*, by *PC.by*, from *PC.for* before time t.

Let the function $verify(msg, key_{pub}, sig)$ denote the return value of using the cryptographic interface $(\texttt{verify}, msg, key_{pub}, sig)$.

Definition 8 *We say that Payment Order PO is acceptable with respect to Payment Certificate PC at time t if the following constraints hold:*

1. *PC was received by machine* $PC.for$ *from machine PC.by.*
2. $verify(PC, PC.by.pub, PC.sig_{by}) = true$*, where PC.by.pub is the public key part of the identifier PC.by.*
3. *PC.path is a suffix of PO.path.*
4. $verify(PO, PC.v, PO.sig_{issuer}) = true$*.*
5. $t \leq \min\{PC.expire, PO.expire - PC.trt\}$*.*
6. *No other PO had been accepted with respect to this PC before, with identical PO.id.*
7. *The net amount that PC.by should pay to* $PC.for$ *does not exceed PC.max, i.e.*

   $$PC.net(PO.amt) \leq PC.max - \sum\{PC.net(po.amt)|po \in ACCEPTED(PC)^t\}$$

8. *For every time interval* (t', t)*, it holds that*

   $$ACCEPTED(PC)^t - ACCEPTED(PC)^{t'} \leq PC.r \cdot (t - t') + PC.b$$

5.3 Issue PC

The instruction for FSP u to issue PC has different parameters, depending the PC flavor.

We recall the PC attributes:

$$PC =< id, by, for, path, expire, trt, max, net, b, r, v, sig_{by} >$$

1. For issuing a prime PC, we use

   $$(\texttt{issue\_pc}, peer, expr, max, net, b, r)$$

   This would yield the following PC:

   $$< id, u, peer, peer||u, expr, 0, max, net, b, r, u, sig_u >$$

2. For issuing a derived PC, from base PC

   $$PC_0 < id_0, by_0, u, path_0, expr_0, trt_0, max_0, net_0, b_0, r_0, v_0, sig_{by_0} >$$

   We use

   $$(\texttt{issue\_pc}, peer, expr, max, net, b, r, PC_{ind})$$

   We give another parameter, the base PC index in u machine database (6.3). This would yield the following PC:

   $$< id, u, peer, peer||path_0, expr, trt_0 + Dalay(u, by_0), max, net, b, r, v_0, sig_u >$$

When FSP issues derived PC, credit of at least $PC.max$, for by_0 is required. Otherwise, if matching PO is redeemed, by_0 will owe $PC.by$ more than the maximal allowed amount. Moreover, if there are other PCs that derived from by_0 PCs, the sum of these PCs max should not exceed the credit given to by_0.

FSP may issue several derived PCs based on the same base PC_0, it is required that every PO that is redeemable against the derived PC, will be redeemable against the base PC_0, so $PC.by$ will not lose money.

Also, for both flavors, the allocation of the Leaky Bucket should be validated. We save the maximum b and r that limits u communication to each peer. Additionally, we save the available b and r of a base PC, and reduce them every time we issue derived PC. The derived PC treatment time, is calculated by the base PC treatment time plus the maximal delay to by_0. So, u will be able to redeem the PO before the base PO would expire.

The implementation is in Algorithms 17, 11.

5.4 Redeem PO

The next stage of the protocol is the redemption. Every FSP is saving balance for its peers, which is changed on redemption and payments. The balance is the amount that this FSP owes to its peer.

The source of the PO is a client of FSP that asks it and deliver it as a payment. When FSP receives a message ($\texttt{redeem}, PO, id, PO_{ind}$) from its peer, it validates that the PO is redeemable against the relevant PC ($PC.id = id$), according to the previous mentioned conditions. If the PO is redeemable, it responds with a signed message ($\texttt{redemption\_certified}, PO_{ind}, PO, time$) and increases the balance for that peer by $PO.amt$. This peer will decrease its balance upon receiving that certificate as well. Of course, if the PO is redeemed against derived PC, the FSP should redeem this PO against the issuer of the base PC.

In this work we consider only one possible redemption path of each PO redemption, which is specified in the PO itself.

The implementation is in Algorithms 13, 14 .

5.5 External Payment

The last stage of the protocol is paying money. The payment is external to our system, and notification about it is provided by the machine administrator. When FSP receives notification ($\texttt{payment\_received}, peer, amt$), where amt is the amount of the payment, it should increase the balance for $peer$ by amt, up to zero (for simplifying the protocol). Also, it should send signed certificate ($\texttt{payment\_certified}, peer, amt, time$) to $peer$. Only after receiving this certificate, $peer$ should decrease its balance as well.

The implementation is in Algorithms 15, 16.

5.6 Communication Messages

Here we give a list of the protocol messages between the peers:

- ($\texttt{given\_credit}, credit$) - This message is to inform the receiver that the sender gave it credit. When received, it evaluated in Algorithm 10.
- ($\texttt{pc\_issued}, PC$) - This message include PC that the sender issued for the receiver.When received, it evaluated in Algorithm 11.
- ($\texttt{redeem}, PO, PC_{id}, PO_{index}$) - This message used to redeem PO against PC with id PC_{id}, it also includes the index of the PO in the sender database, to be referred in the certification. When received, it evaluated in Algorithm 13.
- ($\texttt{redemption\_certified}, PO_{index}, peer, amount, time, signature$) - This message is to certify that the sender accepted the PO that was redeemed by the receiver. When received, it evaluated in Algorithm 14.
- ($\texttt{payment\_certified}, peer, amount, time, signature$) - This message is to certify that the sender received an external payment of $amount$ from the receiver. When received, it evaluated in Algorithm 15.

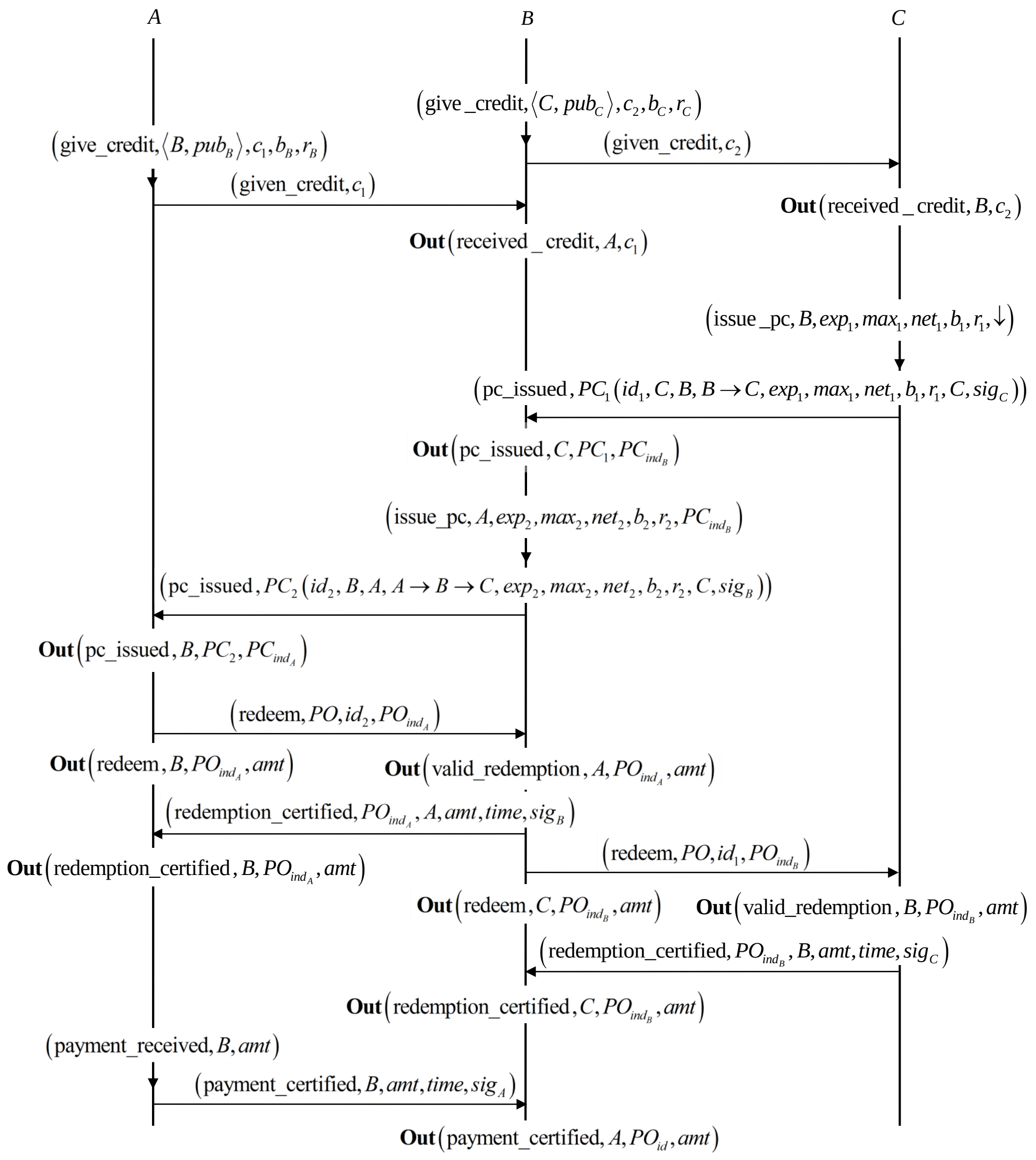
A
B
C
$(\text{give\_credit}, \langle C, pub_C \rangle, c_2, b_C, r_C)$
$(\text{give\_credit}, \langle B, pub_B \rangle, c_1, b_B, r_B)$
$(\text{given\_credit}, c_2)$
$(\text{given\_credit}, c_1)$
$\textbf{Out}(\text{received\_credit}, B, c_2)$
$\textbf{Out}(\text{received\_credit}, A, c_1)$
$(\text{issue\_pc}, B, exp_1, max_1, net_1, b_1, r_1, \downarrow)$
$(\text{pc\_issued}, PC_1(id_1, C, B, B \to C, exp_1, max_1, net_1, b_1, r_1, C, sig_C))$
$\textbf{Out}(\text{pc\_issued}, C, PC_1, PC_{ind_B})$
$(\text{issue\_pc}, A, exp_2, max_2, net_2, b_2, r_2, PC_{ind_B})$
$(\text{pc\_issued}, PC_2(id_2, B, A, A \to B \to C, exp_2, max_2, net_2, b_2, r_2, C, sig_B))$
$\textbf{Out}(\text{pc\_issued}, B, PC_2, PC_{ind_A})$
$(\text{redeem}, PO, id_2, PO_{ind_A})$
$\textbf{Out}(\text{redeem}, B, PO_{ind_A}, amt)$
$\textbf{Out}(\text{valid\_redemption}, A, PO_{ind_A}, amt)$
$(\text{redemption\_certified}, PO_{ind_A}, A, amt, time, sig_B)$
$(\text{redeem}, PO, id_1, PO_{ind_B})$
$\textbf{Out}(\text{redemption\_certified}, B, PO_{ind_A}, amt)$
$\textbf{Out}(\text{redeem}, C, PO_{ind_B}, amt)$
$\textbf{Out}(\text{valid\_redemption}, B, PO_{ind_B}, amt)$
$(\text{redemption\_certified}, PO_{ind_B}, B, amt, time, sig_C)$
$\textbf{Out}(\text{redemption\_certified}, C, PO_{ind_B}, amt)$
$(\text{payment\_received}, B, amt)$
$(\text{payment\_certified}, B, amt, time, sig_A)$
$\textbf{Out}(\text{payment\_certified}, A, PO_{id}, amt)$

Fig. 7. Protocol Overview Illustration

6 Protocol Descriptions

We give pseudo code implementation to the inter-FSP funds transfer protocol in the Algorithms below 6.4. Each algorithm describes evaluation of M^{FSP} machine input interface. Algorithm 9 react upon communication layer input, and calls the relevant function to evaluate it.

6.1 Notations

We briefly sketch the definitions from [3], and map some of the notations to more convenient forms. The symbol $\downarrow$ is the error element available as an addition to the domains and ranges of all functions. We map the notation *null* for it.

Generally, a database D is a countable set of functions, called entries, each over a finite domain called attributes. For an entry $x \in D$, the value in an attribute *att* is written $x.att$. For a predicate *pred* involving attributes, $D[pred]$ means the subset of entries whose attributes fulfill *pred.* If $D[pred]$ contains only one element, we use the same notation for this element. Adding an entry x to D is abbreviated $D :\Leftarrow x$, we map it to D : $\mathbf{insert}(x)$. A database D of machine that owned by u, is denoted by D_u.

The set $\mathcal{INDS}$, isomorphic to $\mathbb{N}$, consecutively numbers all entries in D. Each database has an incremental index $cur\_ind \in \mathcal{INDS}$, that is initialized to zero, and incremented on every entry insertion. The first attribute of the entry is the index, which is set to cur_ind inexplicitly. We use the notation $cur\_ind^{db\_name}$ for the incremental index of the database D^{db_name}.

The index is used as a primary key attribute of the database, i.e., we write $D[i]$ for the selection $D[ind = i]$. We further use the convention that look-ups in D always return the element with the smallest index whose attributes fulfill the queried predicate.

The private key that u machine has is denoted by pks_u^{hnd}, and the public key that u saves for v is denoted by $pks_{u,v}^{hnd}$. We map these notations to $private\_key$, and $public\_key_v$, respectively, for u machine.

6.2 Cryptographic library abstraction

The algorithms are written in pseudo code that supports the abstraction of the cryptographic library implementation, i.e. using handles. An handle is a local name that the machine use for referencing cryptographic terms. We use the set $\mathcal{HNDS}$ to represent the handles.

This mechanism allows the same protocol description to be implemented once with idealized cryptography and once with real cryptography. In the idealized version the handles are local names of Dolev-Yao-style terms, and in the real version, of bit-strings.

For this kind of work, the cryptographic library interface includes the commands:

- **list** forms a list of handles.

- **parse_list** retrieves a handle to the i-th element in the list. It equivalent to the **list_proj** interface in the original notations.
- **store**, **retrieve** inserts and gets data from the cryptographic library using handles.

In the algorithms, the superscript *hnd* on a parameter denotes that it is handle. We use the cryptographic library interfaces only with handle parameters. Thus, we call the function *store* for getting an handle of a normal parameter, before signing, and *retrieve* before database insertions etc.

6.3 Databases

Inter-FSP funds transfer protocol uses databases to maintain the data it needs for running. The first database is D^{FSP}, each entry x in it has the arguments

$$(ind, peer, credit, balance, cert, b, r, t, x)$$

with the following types and meaning for some peer u:

- $x.ind \in \mathcal{INDS}$, index in the database.
- $x.peer$, partner peer id.
- $x.credit \in \mathbb{N}$, the remaining credit for $x.peer$.
- $x.balance \in \mathbb{N}$, the amount that u owes to $x.peer$.
- $x.cert \in \mathbb{N}$, holds the most recent certificate of external payment to $x.peer$. This certificate is received by a message `payment_certified` from $x.peer$.
- $x.b, x.r \in \mathbb{N}$, the maximal leaky bucket parameters that can be given to this peer. These parameters limit the rate that u can send messages to this peer.
- t, x are used for verifying of the Leaky Bucket allocated for sending messages to $x.peer$

Another database is D^{PC} for handling PCs, each entry x in it has the arguments

$$(ind, pc\_id, by, for, path, expire, max, net, b, r, v, sig, t, x, avl, base)$$

with the following types and meaning for some peer u:

- $x.ind \in \mathcal{INDS}$, index in the database.
- $x.pc\_id, pc\_by, for, path, expire, trt, max, net, b, r, v, sig$, PC attributes as they were defined above in Definition 5.
- $x.base$, in case of derived PC, this is the index of the base PC, otherwise it is $null$.
- $x.avl \in \mathbb{N}$, the part of the maximum amount given in the PC, u hasn't used yet.
- t, x are used for verifying of the Leaky Bucket allocated for this PC redemptions, t is the last update time, and x is the last bucket value.

In addition, there is the database D^{PO} for handling POs and certificates. Each entry x in D_u^{PO} has the arguments

$$(ind, id, issuer, path, amt, expire, sig, pc, cert)$$

with the following types and meaning for some peer u:

- $x.ind \in \mathcal{INDS}$, index in the database.
- $x.id, issuer, path, amt, expire, sig$, PO attributes as was defined above.
- $x.pc \in \mathcal{INDS}$, the corresponding PC index in D_u^{PC} of the PO.
- $x.cert \in \mathbb{N}$, holds the certificate of the PO redemption. This certificate is received by a message `redemption_certified`.

6.4 Protocol Algorithms

In the algorithms below we give our pseudo code implementation to the protocol as described in Section 5. An high level overview of the protocol can be found at Fig. 7.

We want to avoid message forgery, so before sending a message the sender always sign, and upon every input from the communication layer the receiver verifies that signature.

For this purpose, we use the function '*sign_send*' (Algorithm 6) for sending every message between peers. This algorithm is also great example of working with handles. We first use the '*store*' and '*list*' function to prepare the parameters for singing (lines 3- 7) and then pack the message and its signature for sending 9. The matching procedure for receiving a message is demonstrated in Algorithm 9, first we split the list of the message and the signature, and then verify them with the public key of the sender.

We denote the running machine owner id by u, and the running machine as M_u^{FSP}.

Algorithm 5: verify_LB(DB, ind)

```
   // updates the pending bucket, and last time of update whenever
      it called
1  $DB[ind].x -= (now() - DB[ind].t)/DB[ind].r$ ;
2  $DB[ind].t = now()$ ;
   // checks if adding one more packet to the pending bucket would
      exceed the max bucket
3  if $DB[ind].x + 1 > DB[ind].b$ then
4      return False;
5  return True;
```

Algorithm 6: sign_send($f, type, parameters$)

```
1 if ($type$ = redeem) ∨ ( // the LB validation is for non-redeem
     messages
2 ($D_u^{FSP}[(peer = f)].ind \neq null$) ∧ ( verify_LB($D_u^{FSP}, D_u^{FSP}[(peer = f)].ind$)
  ) = True) then
3     $type^{hnd} \leftarrow$ store($title$) ;
4     $l^{hnd} \leftarrow$ list($title^{hnd}$) ;
5     foreach $param \in parameters$ do
6         $param^{hnd} \leftarrow$ store($param$) ;
7         add_to_list($l^{hnd}, param^{hnd}$) ;
8     $s^{hnd} \leftarrow$ sign($private\_key, l^{hnd}$) ;
9     $m^{hnd} \leftarrow$ list($l^{hnd}, s^{hnd}$) ;
10    send($f, m^{hnd}$) ;
11    $D_u^{FSP}[(peer = f)].x+ = 1$ ;
```

Algorithm 7: Evaluation of `init_request` instruction in M_u^{FSP}

```
  Input: (init_request, $security\_parameter, communication\_parameters$)
1 $L, R \leftarrow communication\_parameters$;
2 ($private\_key, public\_key_u, id$) ← generate_id($security\_parameter$) ;
3 Output(init_response, $id, public\_key_u$) ;
4 return success;
```

Algorithm 8: Evaluation of `give_credit` instruction in M_u^{FSP}

```
  Input: (give_credit, $f, k, c, b, r$)
1 if ($D_u^{FSP}[(peer = f)].ind = null$) ∧ ($c > 0$) ∧ ($b \geq 0$) ∧ ($r \geq 0$) ∧ ($r < R$)
  then
2     $D_u^{FSP}$ : insert($f, c, 0, null, b, r, now(), 0$) ;
      // attributes are (peer,credit,balance,cert,b,r,t,x)
3     $public\_key_f \leftarrow k$ ;
4     sign_send($f$, given_credit, $c$) ;
5     return success;
6 else
7     return failure;
```

Algorithm 9: Evaluation of inputs from the Communication Layer in M_u^{FSP}

```
Input: $(f, l^{hnd})$
1  $l_j^{hnd} \leftarrow$ parse_list($l^{hnd}, j$) for $j = 1, 2$ ;
2  if verify($l_1^{hnd}, public\_key_f, l_2^{hnd}$) = True then
3      $type^{hnd} \leftarrow$ parse_list($l_1^{hnd}$, 1) ;
4      $type \leftarrow$ retrieve($type^{hnd}$) ;
5      for $i = 1$ to list_length($l_1^{hnd}$) do
6          $param^{hnd} \leftarrow$ parse_list($l_1^{hnd}, i$) ;
7          $param \leftarrow$ retrieve($param^{hnd}$) ;
8          add_to_list($parameter\_list, param$) ;
9      if $type$ = given_credit then
10         evaluate_given_credit($f, parameter\_list$) ;
11     else if $type$ = pc_issued then
12         evaluate_pc_issued($f, parameter\_list, l_2^{hnd}$) ;
13     else if $type$ = redeem then
14         evaluate_redeem($f, parameter\_list$) ;
15     else if $type$ = redemption_certified then
16         evaluate_redemption_certified($f, parameter\_list, l_2^{hnd}$) ;
17     else if $type$ = payment_certified then
18         evaluate_payment_certified($f, parameter\_list, l_2^{hnd}$) ;
```

Algorithm 10: evaluate_given_credit$(f, credit, b, r)$

```
1  Output(received_credit, $credit, f, b, r$) ;
```

Algorithm 11: evaluate_pc_issued(f, pc)

```
1  $(id, by, for, path, expire, trt, max, net, b, r, v, sig) \leftarrow$ parse_list ($pc$);
2  $j := D_u^{PC}[(pc\_id = id) \wedge (pc\_by = by)].ind$ ;
3  if $(D_u^{FSP}[peer = f] \neq null) \wedge (j = null) \wedge (by = f) \wedge (for = u) \wedge($ prefix($u||f, path$)
       = True) $\wedge (expire > now()) \wedge$ // check this is reasonable PC
4  $(net(max) \leq D_u^{FSP}[peer = f].credit) \wedge$ // net amount does not exceed the credit
       for the base pc issuer
5  $(D_u^{FSP}[peer = f].b \geq b) \wedge (D_u^{FSP}[peer = f].r \geq r)$ then
6      $D_u^{PC}$ : insert($pc, now(), 0, max, null$) ;
       // attributes are
           (PC(pc_id,by,path,expire,trt,max,net,b,r,v,sig),t,x,avl,base)
7      $D_u^{FSP}[peer = f].credit -= net(max)$ ;
8      $D_u^{FSP}[peer = f].b -= b$ ;
9      $D_u^{FSP}[peer = f].r -= r$ ;
10     Output((pc_issued, $pc$)) ;
```

Algorithm 12: is_acceptable($po, pc\_id, time$)

```
1  $(po\_id, po\_issuer, po\_path, po\_amt, po\_expire, po\_sig) \leftarrow$ parse_list ($po$);
2  if ($D_u^{PC}[ind = pc\_id].ind! = null$) // pc was issued
3  $\wedge$ verify($po, D_u^{PC}[pc\_id].v, po\_sig$) $=$ True) // po signature verified
4  $\wedge(D_u^{PO}[id = po\_id \wedge issuer = po\_issuer].ind = null)$ // po id is unique
5  $\wedge(D_u^{PC}[pc\_id].net(po\_amt) \leq D_u^{PC}[pc\_id].avl)$ // po net amount does not exceed the
       pc max amount
6  $\wedge$(verify_LB($D_u^{PC}, D_u^{PC}[pc\_id].ind$) $=$ True) // check the redemption rate
7  $\wedge(time \leq po\_expire - D_u^{PC}[pc\_id].trt)$ // we have the treatment time before po
       expiration
8  $\wedge(time \leq D_u^{PC}[pc\_id].expire)$ // pc didn't expired
9  $\wedge$(suffix($D_u^{PC}[pc\_id].path, po\_path$) $=$ True) // po path matches the pc path
10 then
11     return True;
12 return False;
```

Algorithm 13: evaluate_redeem(f, po, pc_id, po_index)

```
1  if is_acceptable($po, pc\_id, now()$) then
2      $D_u^{PO}$ : insert($po\_id, po\_issuer, po\_path, po\_amt, po\_expire, po\_sig, pc\_id, null$) ;
       // attributes are (id,issuer,path,amt,expire,sig,pc,cert)
3      $D_u^{PC}[pc\_id].avl- = D_u^{PC}[pc\_id].net(po\_amt)$ ;
4      $D_u^{PC}[pc\_id].x+ = 1$ ;
5      $D_u^{FSP}[peer = f].balance+ = D_u^{PC}[pc\_id].net(po\_amt)$ ;
6      sign_send($f$, redemption_certified, $po\_index, f, D_u^{PC}[pc\_id].net(po\_amt), now()$) ;
7      Output(valid_redemption, $f, po\_index, D_u^{PC}[pc\_id].net(po\_amt)$) ;
8      $base\_id := D_u^{PC}[pc\_id].base$ ;
9      if $(base\_id \neq null) \wedge (D_u^{PC}[base\_id].ind \neq null)$ then
10         $p := D_u^{PC}[base\_id].by$ ;
11         $D_u^{FSP}[peer = p].balance- = D_u^{PC}[base\_id].net(po\_amt)$ ;
12         $D_u^{PC}[base\_id].x+ = 1$ ;
13         sign_send($p$, redeem, $po, base\_id, cur\_ind^{PO}$) ;
14         Output(redeem, $p, cur\_ind^{PO}, D_u^{PC}[base\_id].net(po\_amt)$) ;
```

Algorithm 14: evaluate_redemption_certified($f, po\_ind, for, amt, time, sig$)

```
1  if
   $(po\_ind \leq cur\_ind^{PO}) \wedge (for = u) \wedge (D_u^{PO}[po\_ind].amt = amt) \wedge (now() - time \leq Delay(f, u))$
   then
2      $D_u^{PO}[po\_ind].cert :=$ list(redemption_certified,$f,po\_ind,for,amt,time,sig$) ;
3      Output(redemption_certified, $f, po\_ind, amt$) ;
```

Algorithm 15: evaluate_payment_certified($f, for, amt, time, sig$)

```
1 if $(for = u) \wedge (D_u^{FSP}[peer = f].balance - amt \geq 0))$ then
2     $D_u^{FSP}[peer = f].balance- = amt$ ;
3     $D_u^{FSP}[peer = f].cert :=$ list(payment_certified,$f$,$for$,$amt$,$time$,$sig$) ;
4     Output(payment_certified, $f, amt$) ;
```

Algorithm 16: Evaluation of `payment_received` instruction in M_u^{FSP}

```
Input: (payment_received, $f, amt$)
1 if $D_u^{FSP}[peer = f].balance + amt \leq 0$ then
2     $D_u^{FSP}[peer = f].balance+ = amt$ ;
3     sign_send($f$, payment_certified, $f, amt, now()$) ;
4     return success;
5 else
6     return failure;
```

7 Security Analysis

7.1 The cryptographic library

The Inter-FSP Funds Transfer Protocol uses cryptographic interfaces for signing messages with private key, verifying such signatures with public key. The security analysis below, assumes that the interfaces are implemented by ideal The Dolev-Yao cryptographic library, which is defined by Backes et al. [3],and denoted by:

$$Sys^{cry,id}$$

Dolev-Yao model [8] represents cryptography as term algebras, and thus, signature will be verified with some public key, if and only if the message was signed with the matching private key. So, the ideal library saves for each signing operation it performs, the tuple ($message, key, signature$), and returns an handle that represents *signature* internally. The verification operation is done by searching for a tuple that matches the input. If such tuple exists it returns true, otherwise it returns false.

Of course, we would like the Inter-FSP Funds Transfer Protocol to be provably secure, with negligible probability with respect to given security parameter, when it uses real cryptographic library, that uses real cryptographic calculations, which is defined in [3] and denoted by:

$$Sys^{cry,real}$$

7.2 The Theorems

Before presenting the theorems, we repeat the definition of *secure machine with respect to Valid_CommLog and Valid_ProtocolLog* from Section 3.

Algorithm 17: Evaluation of `issue_pc` instruction in M_u^{FSP}

```
Input: (issue_pc, $f, expr, max, net, x, b, r, base\_ind$)
1  if ($base\_ind = null$) then
       // prime PC
2      $path \leftarrow$ list($f,u$) ;
3      $pc \leftarrow$ list($cur\_ind^{PC}, u, f, path, expr, 0, max, net, b - r * Delay(u, f), r, public\_key_u$) ;
4      $sig \leftarrow$sign($private\_key, pc$) ;
5      $pc \leftarrow$ list($pc, sig$) ;
6      $D_u^{PC}$ : insert($pc, now(), 0, max, null$) ;
       // attributes are
          (PC(pc_id,by,for,path,expire,trt,max,net,b,r,v,sig),t,x,avl,base)
7      sign_send($f$, pc_issued, $pc$) ;
8      return success;
9  else
       // derived PC
10     $p \leftarrow D_u^{PC}[base\_ind].by$ ;
11     if ($base\_ind \leq cur\_ind^{PC} \wedge p \neq u$) // base PC exists
12     $\wedge(expr \leq D_u^{PC}[base\_ind].expr - Delay(u, p))$ // pc expires before its base PC
13     $\wedge(D_u^{PC}[base\_ind].net(max) \leq D_u^{PC}[base\_ind].avl)$ // max amount does not exceed
          the base PC
14     $\wedge(b \leq D_u^{PC}[base\_ind].b) \wedge (r \leq D_u^{PC}[base\_ind].r)$// committed rate not higher than
          the base PC
15     then
16         $D_u^{PC}[base\_ind].avl- = D_u^{PC}[base\_ind].net(max)$ ;
17         $D_u^{PC}[base\_ind].b- = b$ ;
18         $D_u^{PC}[base\_ind].r- = r$ ;
19         $path \leftarrow$ list($f, D_u^{PC}[base\_ind].path$) ;
20         $trt \leftarrow D_u^{PC}[base\_ind].trt + Delay(u, p)$ ;
21         $pc \leftarrow$ list($cur\_ind^{PC}, u, f, path, expr, trt, max, net, b, r, D_u^{PC}[base\_ind].v$) ;
22         $sig \leftarrow$sign($private\_key, pc$) ;
23         $pc \leftarrow$ list($pc, sig$) ;
24         $D_u^{PC}$ : insert($pc, now(), 0, max, base\_ind$) ;
           // attributes are
              (PC(pc_id,by,for,path,expire,trt,max,net,b,r,v,sig),t,x,avl,base)
25         sign_send($f$, pc_issued, $pc$) ;
26         return success;
27     else
28         return failure;
```

Algorithm 18: Evaluation of `pc_timeout` instruction in M_u^{FSP}

```
Input: (pc_timeout, $pc\_id$)
if $D_u^{PC}[id = pc\_id].expr > now()$ then
    $base\_id = D_u^{PC}[id = pc\_id].base$ ;
    if $D_u^{PC}[id = pc\_id].by = u \wedge (base\_id \neq null)$ then
        $D_u^{PC}[base\_id].avl += D_u^{PC}[id = pc\_id].net(D_u^{PC}[id = pc\_id].max)$ ;
        $D_u^{PC}[base\_id].b += b$ ;
        $D_u^{PC}[base\_id].r += r$ ;
    else
        $f \leftarrow D_u^{PC}[id = pc\_id].by$ ;
        $D_u^{FSP}[peer = f].credit += D_u^{PC}[id = pc\_id].net(D_u^{PC}[id = pc\_id].max)$ ;
        $D_u^{FSP}[peer = f].b += D_u^{PC}[id = pc\_id].b$ ;
        $D_u^{FSP}[peer = f].r += D_u^{PC}[id = pc\_id].r$ ;
    return success;
else
    return failure;
```

Algorithm 19: Evaluation of `issue_po` instruction in M_u^{FSP}

```
Input: (issue_po, $path, amt, expire$)
$po^{hnd} \leftarrow$ list($cur\_ind^{PO}, u, path, amt, expire$) ;
$sig^{hnd} \leftarrow$ sign($private\_key, po^{hnd}$) ;
Output($po^{hnd}, sig^{hnd}$) ;
return success;
```

Algorithm 20: Evaluation of `po_received` instruction in M_u^{FSP}

```
Input: (po_received, $po, pc\_id$)
$(po\_id, po\_issuer, po\_path, po\_amt, po\_expire, po\_sig) \leftarrow$ parse_list ($po$);
$p := D_u^{PC}[pc\_id].by$ ;
if $(p \neq null) \wedge (p \neq u) \wedge$ is_acceptable($po, pc\_id, now() + Delay(u, p)$) then
    $D_u^{PO}$ : insert($po\_id, po\_issuer, po\_path, po\_amt, po\_expire, po\_sig, pc\_id, null$) ;
    // attributes are (id,issuer,path,amt,expire,sig,pc,cert)
    $D_u^{FSP}[peer = p].balance -= D_u^{PC}[pc\_id].net(po\_amt)$ ;
    $D_u^{PC}[pc\_id].avl -= D_u^{PC}[pc\_id].net(po\_amt)$;
    $D_u^{PC}[pc\_id].x += 1$ ;
    sign_send($p$, redeem, $po, pc\_id, cur\_ind^{PO}$) ;
    Output(redeem, $p, po\_index, D_u^{PC}[pc\_id].net(po\_amt)$) ;
    return success;
else
    return failure;
```

Definition 1 *We say that machine $\mathcal{M}$ is a* secure machine *with respect to $Valid\_CommLog$ and $Valid\_ProtocolLog$, if for every polynomial adversary $\mathcal{A}$, the probability*

$$Pr[Game(\mathcal{M}, Valid\_CommLog, Valid\_ProtocolLog, \mathcal{A}, 1^k) = False]$$

is negligible with respect to the security parameter 1^k.

Theorem 1. *The proposed Inter-FSP Funds Transfer Protocol with an ideal cryptogaphic library is a* secure $\mathcal{M}^{FSP}$ machine *with respect to $FSP\_Valid\_ProtocolLog$ and $LR\_FIFO\_Valid\_CommLog$.*

Theorem 2. *The proposed Inter-FSP Funds Transfer Protocol with a real cryptogaphic library is a* secure $\mathcal{M}^{FSP}$ machine *with respect to $FSP\_Valid\_ProtocolLog$ and $LR\_FIFO\_Valid\_CommLog$.*

Proof. Backes et al. proved in ([3], Theorem 1. Security of Cryptographic Library) the following claim: *The real cryptographic library is as secure as the ideal cryptographic library, so that protocols proved on the basis of the deterministic, Dolev-Yao-like ideal library can be safely implemented with the real cryptographic library.*

With their notation, they proved that:

$$Sys^{cry,real} \geq Sys^{cry,id}$$

That is, any polynomial attacker that can break the system when it uses the real cryptographic library, with non-negligible probability, can break the system when it uses the ideal one.

According to Theorem 1, Inter-FSP Funds Transfer Protocol is secure with $Sys^{cry,id}$. So, it follows from Backes et al. theorem, that Inter-FSP Funds Transfer Protocol is secure with $Sys^{cry,real}$.

■

In the following sections we prove Theorem 1. In Section 7.3 we present claims that help to prove Theorem 1. In Section 7.4 we show that if these claims hold, Theorem 1 holds. In Section 7.5 we give the proofs of the claims.

7.3 Claims

We claim that the balance that is saved in the machine is equal to the balance that is saved in $FSP\_Valid\_ProtocolLog$.

Claim 1 *When running $FSP\_Valid\_ProtocolLog$ during the game, the value of $Balance[u][v]$ is equivalent to the value of $D_u^{FSP}[peer = v].balance$.*

We claim that $Delay(u, v)$ bounds the communication delay of redemption request from u to v.

Claim 2 *If $LR\_FIFO\_Valid\_CommLog(CommLog) = True$, then if an honest party u sends a* `redeem` *message to v at time t, it should be received by v at time t_2 s.t. $t \leq t_2 \leq t + Delay(u, v)$.*

We claim that the credit bounds the negative balance.

Definition 9 *Let $Credit(u, v)$ denote the credit c given by the first instruction*

$$(\texttt{give\_credit}, v, c, b, r)$$

from u upper layer, where $c > 0$.

Claim 3 *For every honest peer u, at every step of running the game,*

$$Credit(u, v) \geq -D_u^{FSP}[peer = v].balance$$

We claim that an honest peer validates redemption request with acceptable PO.

Claim 4 *If an honest peer u received a payment order PO from its peer v at time t, and PO was acceptable, at that time t, with respect to some payment certificate PC that u issued for v, then u should output*

$$(\texttt{valid\_redemption}, v, PO.id, PC.net(PO.amt))$$

immediately.

We claim that two honest peers store the same PC attributes of PC's that they issued for each other.

Claim 5 *Let x be a record in database D_u^{PC} of an honest peer u, and let v denote $x.by$. If v is honest, there is a record y in D_v^{PC} s.t.:*

$$\forall attribute \in \{pc\_id, by, for, path, expire, trt, max, net, b, r, v, sig\}$$

it holds that:

$$x.attribute = y.attribute$$

We claim that an honest peer can check if a PO will be acceptable in other peer.

Claim 6 *Let u be an honest party, and let x be a record in D_u^{PC}, where $x.by$ is honest too. Also, let v denote $x.by$, and let po be some payment order.*

For any time t_1, if u runs the function

$$is\_acceptable(po, x.id, t_1 + Delay(u, v))$$

at time t_1 and it returns true, if v runs

$$is\_acceptable(po, x.id, t_2)$$

at t_2 s.t. $t_1 \leq t_2 \leq t_1 + Delay(u, v)$, it will return true as well.

We claim that every PO that is acceptable with respect to derived PC (Definition 6), will be acceptable with respect to the base PC.

Claim 7 *Let pc_derived be a derived payment certificate that an honest peer u issued based on payment certificate pc_base from p. Every payment order that is acceptable with respect to pc_derived at time t, will be acceptable with respect to pc_base at any time $t_2 | t \leq t_2 \leq t + Delay(u, p)$.*

7.4 Proof of Theorem 1 based on the claims

Consider the *FSP_Valid_ProtocolLog* procedure in Algorithm 4. We need to show that for every run of the game, if

$$LR\_FIFO\_Valid\_CommLog(CommLog) = True$$

then

$$FSP\_Valid\_ProtocolLog(ProtocolLog) = True$$

i.e. the conditions at lines 19 and 27 are never satisfied.

We start with the condition at line 27, the condition is:

$$Credit[u][v] < -Balance[u][v]$$

It is easy to see that $Credit[u][v]$ is equivalent to $Credit(u, v)$. In addition, Claim 1 says that the $D_u^{FSP}[peer = v].balance$ that is saved in the machine is equal to the $Balance[u][v]$ that is saved in $FSP\_Valid\_ProtocolLog$. So, it follows from Claim 3 which says $Credit(u, v) \geq -D_u^{FSP}[peer = v].balance$, that this condition will never be satisfied.

Now, we show that the condition at line 19, is never satisfied. For this, we need to show that for every record

$$(u, t_1, \texttt{redeem}, v, id, amt)$$

in the *ProtocolLog*, where v is honest, there is a matching record

$$(v, t_2, \texttt{valid\_redemption}, u, id, amt)$$

or a record

$$(\texttt{finish\_game}, t_2)$$

where $t_2 - t_1 \leq Delay(u, v)$.

Recall that by definition, only honest parties write to *ProtocoLog*, and the first attribute of a record in *ProtocolLog* is the writing peer identifier.

Thus, a record $(u, t_1, \texttt{redeem}, v, id, amt)$ may exist in *ProtocolLog*, only if an honest peer u outputs

$$(\texttt{redeem}, v, id, amt)$$

at time t_1. Such output can occur only in Algorithm 20. at line 9 and in Algorithm 13. at line 14:

1. Algorithm 20. line 9.
Algorithm 20 is the evaluation of ($\texttt{po\_received}, po, pc\_id$) instruction, of payment order that received from the upper layer. To reach line 9, the conditions at line 3 must be satisfied, in particular it must hold that
$$is\_acceptable(po, pc\_id, t_1 + Delay(u, v)) = true$$
Also, a message ($\texttt{redeem}, po, pc\_id, po\_ind$) is sent to v (line 8), and should be received in v at time t_2 s.t. $t_2 \leq t_1 + Delay(u, v)$, according to Claim 2. Of course, if the game ended before the redeem message arrived we get the line
$$(\texttt{finish\_game}, t_2)$$
s.t. $t_2 \leq t_1 + Delay(u, v)$ in $ProtocolLog$ as required.
If v is honest, its machine should evaluate such input using Algorithm 13. Thus, v should output ($\texttt{valid\_redemption}, u, id, amt$) at time t_2 (line 9), if the function
$$is\_acceptable(po, pc\_id, t_2)$$
(line 1) returns true. As it was mentioned before, the condition in Algorithm 20 at line 3 must be satisfied in u. So, there is a record x in D_u^{PC} where $x.id = pc\_id$, and $x.by = v$. According to Claim 6, in this case, if
$$is\_acceptable(po, pc\_id, t_1 + Delay(u, v))$$
returns true in u, then
$$is\_acceptable(po, pc\_id, t_2)$$
must return true in v, and v should output ($\texttt{valid\_redemption}, u, id, amt$). This output will lead to the required ($v, t_2, \texttt{valid\_redemption}, u, id, amt$) record in $ProtocolLog$.
2. Algorithm 13. line 14.
To reach line 14, the condition at line 1 must be satisfied, i.e the input payment order po was acceptable at time t_1 with respect to the payment certificate PC that is stored in the record $D_u^{PC}[id = pc\_id]$. Also, to reach line 14, the condition at line 9 must be satisfied. That is, the payment certificate PC, was derived from another payment certificate PC_base.
Additionally, line 13 must be reached, and u should send a redeem message to v, which is $PC\_base.by$.
According to Claim 2, this message will arrive to v no later than $t_1 + Delay(u, v)$.
Of course, if the game ended before the redeem message arrived we get the line
$$(\texttt{finish\_game}, t_2)$$
s.t. $t_2 \leq t_1 + Delay(u, v)$ in $ProtocolLog$ as required.
According to Claim 7. every payment order that is acceptable with respect to a derived PC at time t_1, is acceptable with respect to the base PC at time $t_1 + Delay(u, v)$.

Also, according to Claim 4, if v is honest, it should output immediately

$$(\texttt{valid\_redemption}, u, id, amt)$$

when it receives the redemption message from u, i.e. at time t_2 s.t. $t_2 \leq t_1 + Delay(u, v)$.
This output will lead to the required $(v, t_2, \texttt{valid\_redemption}, u, id, amt)$ record in *ProtocolLog*.

■

7.5 Proof of the Claims

For proving the Claims, we first add sub-claims and prove them.

Sub-Claim 1 *Consider an honest party u. There is at most one record per peer in D_u^{FSP}.*

Proof. The only insertion of record to D_u^{FSP}, is in Algorithm 8. line 2, and the condition at line 1 ensures that there is no other record with the same peer in D_u^{FSP}. ■

Sub-Claim 2 *Consider an honest party u. If u sends redemption message to its peer v, then u must have exactly one record $D_u^{FSP}[peer = v]$.*

Proof. The only places that u can send a `redeem` message are in Algorithm 20 and in Algorithm 13. In both places we check first that a payment certificate from v exists in D_u^{PC} (at line 3 and at line 9, respectively). Such payment certificate may be inserted to D_u^{PC} only in Algorithm 11 at line 6. To reach this insertion, the condition at line 3 must be satisfied, and particularly, it must hold that $D_u^{FSP}[peer = v] \neq null$. According to Sub-Claim 1 there is at most one record $D_u^{FSP}[peer = v]$, so u must have exactly one record $D_u^{FSP}[peer = v]$. ■

Sub-Claim 3 *An honest party will never send redemption against payment certificate PC unless the rate meets the LB limitation with $PC.b, PC.r$ as parameters.*

Proof. The only places that u can send a redemption are in Algorithm 20 and in Algorithm 13. In Algorithm 20, u sends the redemption only if the function *is_acceptable* returns true. The function *is_acceptable* is implemented in Algorithm 12, and it returns true only if the condition at line 6 is satisfied. This condition uses the function $verify\_LB(DB, ind)$ (Algorithm 5). The function $verify\_LB(DB, ind)$ verifies if sending a single message at the time of calling it, is valid with respect to the LB model with the parameters $DB[ind].b, DB[ind].r$. It uses the attribute $DB[ind].x$ which represent the size of the bucket of the pending messages on this link, and updates $DB[ind].x$ according to the attribute $DB[ind].t$, which is the last update time.

The record that is passed to $verify\_LB(DB, ind)$ is the record of PC in D_u^{PC}, i.e. $D_u^{PC}[id = PC.id]$, so the LB validation is against $PC.b, PC.r$. Also, the x attribute of this record is incremented by each request (Algorithm 13 line 4, and Algorithm 20 line 7).

In Algorithm 13, u sends a redemption of a payment order that was accepted with respect to derived payment certificate. The redemption is sent to the issuer of the base payment certificate *base_pc*. We show that the sum of b, r parameter among all derived payment certificates from *base_pc*, doesn't exceed $pc\_base.b, pc\_base.r$, respectively. Issuing a derived payment certificate is done in Algorithm 17 there the derived payment certificate b, r attributes are checked, in line 14, against the attributes b, r in *pc_base* record in D_u^{PC}. These attributes are initialized to $pc\_base.b, pc\_base.r$ when *pc_base* was received, in Algorithm 11 at line 6. For every derived PC from *pc_base* these attributes are reduced by the b, r of the derived PC at lines 17 and 18 (When the derived PC is timed out these attributes are increased by the same values Algorithm 18 lines 5,6).

This way the derived payment certificates rates are pre-allocated from the total LB parameters of the base payment certificate, and the redemption of payment orders that are received from the upper layer are verified against the reminder of the LB parameters. ■

Sub-Claim 4 *The net amount of redemptions of an honest peer against some PC, is bounded by the net amount of the maximal amount of the PC.*

Proof. An honest peer u, saves in *avl* attribute of a record x in D_u^{PC} the remaining amount of using the payment certificate that is stored in this record. This attribute is initialized to the $x.net(x.max)$ when the received payment certificate is stored (Algorithm 11. line 6).

Sending redemption can be done only in Algorithm 20, or in Algorithm 13.

In Algorithm 20, u checks that the net amount of the redemption does not exceed this *avl*, using the *is_acceptable* function (Algorithm 12.line 5), and updates it at line 6.

In Algorithm 13, u would redeem payment certificate only if it is acceptable with respect to some derived payment certificate, and it would redeem it to the issuer of the base payment certificate automatically. So, the check and the update of *avl*, must be done when u issues derived PC.

We look into Algorithm 17, that describes the evaluation of `issue_pc` instruction. Issuing of derived payment certificate *pc_derived*, that is based on payment certificate *pc_base*, is done by u only if the condition at line 13 is satisfied. This condition checks the attribute *avl* in the record $D_u^{PC}[id = pc\_base.id]$ and together with the update at line 16 (and with the update of timeout PC in Algorithm 18, line 4), it bounds the total maximal net amount of all derived payment certificates that are based on *pc_base*, to the initial value of this *avl* attribute, which is the net amount of the maximal amount of *pc_base* (Algorithm 11. line 6). ■

Sub-Claim 5 *When running Algorithm 17. which evaluates issuing of derived PC, the condition that ends at line 15 is satisfied if and only if every acceptable*

PO with respect to the derived PC at some time t, is acceptable with respect to the base PC at any time $t_2|t \leq t_2 \leq t + Delay(u,p)$, where p is the issuer of the base PC.

Proof. Let *pc_derived* be the derived PC, issued by an honest peer u for some peer f, using Algorithm 17, i.e. the condition that ends at line 15 was satisfied. Let *pc_base* be the base PC of *pc_derived*, issued by other peer p. Also, let *po* be an acceptable payment order with respect to *pc_derived* at some time t.

We repeat Definition 8, that specifies the constrains that should hold for payment order PO to be acceptable with respect to payment certificate PC, and show that the claim holds, that is, *po* is acceptable with respect to *pc_base* at time $t + Delay(u,p)$, by referring to lines in Algorithm 17. For readability, we rewrite the definition and add our proof of each part inline:

1. *PC was received by machine PC.for from machine PC.by.*
   This is checked in line 11, a record with x s.t. $x.by \neq u$ may exist in D_u^{PC} only if it was inserted in Algorithm 11 at line 6, that is, (`PC_issued`, *pc_base*) message was received in u, from the issuer p.
2. $verify(PC, PC.by.pub, PC.sig_{by}) = true$, where $PC.by.pub$ is the public key part of the identifier $PC.by$.
   As it was mentioned above, *pc_base* was received from the communication layer as (`PC_issued`, *pc_base*) message from p. The verification of the signature is done in Algorithm 9 at line 2 before the evaluation of such input in Algorithm 11.
3. $PC.path$ is a suffix of $PO.path$.
   If *po* is acceptable with respect to *pc_derived*, then *pc_derived.path* must be a suffix of *po.path*. According to line 19, *pc_base.path* is a suffix of *pc_derived.path*. So, *pc_base.path* is a suffix of *po.path*.
4. $verify(PO, PC.v, PO.sig_{issuer}) = true.$
   If *po* is acceptable with respect to *pc_derive* it must satisfy this, since $pc\_derived.v = pc\_base.v$ according to line 21.
5. $t \leq \min\{PC.expire, PO.expire - PC.trt\}$.
   If *po* was accepted with respect to *pc_derived* at time t, that is $t \leq po.expr - pc\_derived.trt$ and $t \leq pc\_derived.expr$.
   In line 12 it is checked that $pc\_derived.expr \leq pc\_base.expr - Delay(u,p)$.
   So, we get that $t + Delay(u,p) \leq pc\_base.expr$.
   Also, according to line 20, $pc\_derived.trt = pc\_base.trt + Delay(u,p)$.
   So, if $t \leq po.expire - pc\_derived.trt$, then $t + Delay(u,p) \leq po.expire - pc\_base.trt$.
6. *No other PO had been accepted with respect to this PC before, with identical PO.id.*
   Of course, *po* is acceptable only once. Every PO has specific redemption path, so if *po* was found unique in u, it will be found unique in p as well, since no other peer can redeem such PO to p.
7. *The net amount that PC.by should pay to PC.for does not exceed PC.max, i.e.*

   $$PC.net(PO.amt) \leq PC.max - \sum\{PC.net(po.amt)|po \in ACCEPTED(PC)^t\}$$

This is checked in line 13, with the attribute avl in pc_base record in D_u^{PC}. According to Sub-Claim 4, this condition bounds the net amount of all redemption against pc_base, including redemptions of payment orders that was accepted with respect to derived PC's from pc_base.

8. *For every time interval* (t', t)*, it holds that*

$$ACCEPTED(PC)^t - ACCEPTED(PC)^{t'} \leq PC.r \cdot (t - t') + PC.b$$

We first show that the sum of b, r parameter among all derived payment certificates from $base\_pc$, doesn't exceed $pc\_base.b, pc\_base.r$, respectively. This is checked in line 14, with the attributes r, b in pc_base record in D_u^{PC}. These attributes are initialized to $pc\_base.b, pc\_base.r$ when pc_base was received, in Algorithm 11 at line 6. For every derived PC from pc_base these attributes are reduced by the b, r of the derived PC at lines 17 and 18 (When the derived PC is timed out these attributes are increased by the same values Algorithm 18 lines 5,6).
This way, if po was acceptable with respect to $pc\_derived$, it was LB verified with $pc\_derived.r, pc\_derived.b$ which together with all derived payment certificates are lower than $pc\_base.r, pc\_base.b$. Thus, po must be LB verified with $pc\_base.r, pc\_base.b$.

So, according to Definition 8, the claim holds. ■

Sub-Claim 6 *The return value of the function* $is\_acceptable(PO, pc\_id, t)$*, where run by an honest peer u at time t, is true if and only if the payment order PO is acceptable at time t with respect to the payment certificate PC which is stored in* $D_u^{PC}[id = pc\_id]$.

Proof. The function $is\_acceptable$ is implemented Algorithm 12. and it returns true only if the condition that ends at line 9 is satisfied.

We repeat Definition 8, that specifies the constrains that should hold for payment order PO to be acceptable with respect to payment certificate PC, and show for each constrain that it holds, by referring to lines in Algorithm 12. For readability, we rewrite the definition and add our proof of each part inline:

1. *PC was received by machine* $PC.for$ *from machine* $PC.by$.
This is checked at line 2, the condition is satisfied only if D_u^{PC} contains a record x, where $x.id = pc\_id$. If $x.by = u$, such record insertion, can take place only in Algorithm 17 at line 6 or at line 24 in case of issuing prime PC, or derived PC, respectively. Right after each of these lines (Algorithm 17 lines 7, 25), the payment certificate PC is sent to $x.for$, which is $PC.for$. Otherwise, if $x.by \neq u$, such record insertion can take place only in Algorithm 11. that evaluates receiving of payment certificate from the communication layer, at line 6. According to the condition at line 3, this $x.by$ is the sender of the message, and $x.for = u$.

2. $verify(PC, PC.by.pub, PC.sig_{by}) = true$, where $PC.by.pub$ is the public key part of the identifier $PC.by$.
   If the issuer of PC is u itself, it should sign PC and store the signature in $PC.sig$ in Algorithm 17 lines 7, 25.
   Otherwise, if PC was received as a communication input, u must verify the signature of the sender on the PC, in Algorithm 2. before evaluating it.
3. $PC.path$ is a suffix of $PO.path$.
   This is checked in line 9.
4. $verify(PO, PC.v, PO.sig_{issuer}) = true$.
   This is checked in line 3.
5. $t \leq \min\{PC.expire, PO.expire - PC.trt\}$.
   This is checked in lines 8 and 7.
6. *No other PO had been accepted with respect to this PC before, with identical* $PO.id$.
   This is checked in line 4, this condition checks that there is no record in D_v^{PO} with id attribute that equals to $PO.id$ from the same issuer, since the payment order id attribute is unique per issuer (Algorithm 19).
   Insertion of a payment order record to D^{PO} can take place only after successful check of $is\_acceptable()$ with this payment order (Algorithm 13. line 2 , Algorithm 20. line 4). So, every payment order can pass $is\_acceptable()$ only one time per machine, when sending or receiving redemption.
7. *The net amount that PC.by should pay to PC.for does not exceed PC.max, i.e.*

   $$PC.net(PO.amt) \leq PC.max - \sum\{PC.net(po.amt) | po \in ACCEPTED(PC)^t\}$$

   This is checked in line 5, using the attribute $x.avl$. According to Sub-Claim 4, this condition bounds the net amount of all redemption against PC.
8. *For every time interval* (t', t)*, it holds that*

   $$ACCEPTED(PC)^t - ACCEPTED(PC)^{t'} \leq PC.r \cdot (t - t') + PC.b$$

   This is checked in line 6, that uses the function $verify\_LB(DB, ind)$ (Algorithm 5). The function $verify\_LB(DB, ind)$ verifies if sending a single message at the time of calling it, is valid with respect to the LB model with the parameters $DB[ind].b, DB[ind].r$. It uses the attribute $DB[ind].x$ which represent the size of the bucket of the pending messages on this link, and updates $DB[ind].x$ according to the attribute $DB[ind].t$, which is the last update time.
   The b, r parameters of the LB verification are the $PC.b, PC.r$ parameters. Also, the $x.x$ incremented by each request (Algorithm 13 line 4, and Algorithm 20 line 7).

■

Proof of Claim 1

Claim 1 *When running $FSP\_Valid\_ProtocolLog$ (Algorithm 4.) during the game, the value of $Balance[u][v]$ is equivalent to the value of $D_u^{FSP}[peer = v].balance$.*

Proof. Consider an honest party u that runs the Inter-FSP Funds Transfer Protocol. According to Sub-Claim 1, there is at most one record per peer in D_u^{FSP}. $D_u^{FSP}[peer = v].balance$ is initialized to zero in Algorithm 8 at line 2, as a result of instruction

$$(\texttt{give\_credit}, v, k, c, b, r)$$

to machine u, where this is the first give_credit instruction for the peer v and $(c > 0) \wedge (b \geq 0) \wedge (r \geq 0) \wedge (r < R)$. Clearly, such instruction will lead to the following line in $ProtocolLog$

$$(u, t, \texttt{give\_credit}, v, k, c, b, r)$$

and when $FSP\_Valid\_ProtocolLog$ (Algorithm 4.) reaches this line in the log, the conditions at lines 5 - 11 will not be satisfied, and thus, at line 14 $Balance[u][v]$ will be initialized to zero as well.

We list below all the places that $D_u^{FSP}[peer = v].balance$ is changed:

1. Algorithm 13 at line 5.
2. Algorithm 13 at line 11.
3. Algorithm 20 at line 5.
4. Algorithm 16 at line 2.
5. Algorithm 15 at line 2.

We prove by induction that $D_u^{FSP}[peer = v].balance$ is equivalent to the value of $Balance[u][v]$, after each possible change according to this list.

1. Added amount amt to $D_u^{FSP}[peer = v].balance$ in Algorithm 13 at line 5. Right after this line, at line 7, u should output

   $$(\texttt{valid\_redemption}, v, po\_ind, amt)$$

   Such output will lead to the following line in $ProtocolLog$:

   $$(u, t, \texttt{valid\_redemption}, v, po\_ind, amt)$$

   and when $FSP\_Valid\_ProtocolLog$ (Algorithm 4.) reaches this line in the log, it will reach line 16 and add amt to $Balance[u][v]$ as well.
2. Subtracted amount amt from $D_u^{FSP}[peer = v].balance$ in Algorithm 13 at line 11.
   Right after this line, at line 14, u should output

   $$(\texttt{redeem}, v, po\_ind, amt)$$

Such output will lead to the following line in *ProtocolLog*:

$$(u, t, \texttt{redeem}, v, po\_ind, amt)$$

and when $FSP\_Valid\_ProtocolLog$ (Algorithm 4.) reaches this line in the log, it will reach line 18 and subtract *amt* from $Balance[u][v]$ as well.

3. Subtracted amount *amt* from $D_u^{FSP}[peer = v].balance$ in Algorithm 20 at line 5.
Right after this line, at line 9, u should output

$$(\texttt{redeem}, v, po\_ind, amt)$$

Such output will lead to the following line in *ProtocolLog*:

$$(u, t, \texttt{redeem}, v, po\_ind, amt)$$

and when $FSP\_Valid\_ProtocolLog$ (Algorithm 4.) reaches this line in the log, it will reach line 18 and subtract *amt* from $Balance[u][v]$ as well.

4. Added amount *amt* to $D_u^{FSP}[peer = v].balance$ in Algorithm 16 at line 2. This Algorithm is called as a result of instruction

$$(\texttt{payment\_received}, v, amt)$$

where $D_u^{FSP}[peer = v].balance + amt \leq 0$. Such instruction will lead to the following line in *ProtocolLog*:

$$(u, t, \texttt{payment\_received}, v, amt)$$

and when $FSP\_Valid\_ProtocolLog$ (Algorithm 4.) reaches this line in the log, it will reach line 24 and add *amt* to $Balance[u][v]$ as well.

5. Subtracted amount *amt* from $D_u^{FSP}[peer = v].balance$ in Algorithm 15 at line 2.
Right after this line, at line 4, u should output

$$(\texttt{payment\_certified}, v, amt)$$

Such output will lead to the following line in *ProtocolLog*:

$$(u, t, \texttt{payment\_certified}, v, amt)$$

and when $FSP\_Valid\_ProtocolLog$ (Algorithm 4.) reaches this line in the log, it will reach line 26 and subtract *amt* from $Balance[u][v]$ as well.

■

Proof of Claim 2

Claim 2 *If $LR\_FIFO\_Valid\_CommLog(CommLog) = True$, then if an honest party u sends a* `redeem` *message to v at time t, it should receive by v at time t_2 s.t. $t \leq t_2 \leq t + Delay(u, v)$.*

Proof. Recall that $Delay(u, v)$ (Definition 3) bounds the communication delay between u to v, as long as the underlying communication layer guarantees to serve sending requests in some service rate according to the LR server model with the parameters L, R. This is given by the assumption of

$$LR\_FIFO\_Valid\_CommLog(CommLog) = True$$

In addition, the sending rate from u to v must be limited by the LB model validation with the parameters b, r, given by the instruction:

$$(\texttt{give\_credit}, v, k, c, b, r)$$

in u, where $(b \geq 0) \wedge (r \geq 0) \wedge (r < R)$.

We need to show that an honest party sending rate meets this limitation. For this purpose we maintain the attributes $D_u^{FSP}[peer = v].b, D_u^{FSP}[peer = v].r$. According to Sub-Claim 2, if u sends redemptions to v, there is exactly one record $D_u^{FSP}[peer = v]$. The attributes $D_u^{FSP}[peer = v].b, D_u^{FSP}[peer = v].r$ are initialized when u inserts the record $D_u^{FSP}[peer = v]$, at line 2 in Algorithm 8, which evaluates the

$$(\texttt{give\_credit}, v, k, c, b, r)$$

instruction. The insertion of the record, according to the condition at line 1, is performed only if $(b \geq 0) \wedge (r \geq 0) \wedge (r < R)$ and the values of $D_u^{FSP}[peer = v].b, D_u^{FSP}[peer = v].r$ are set to the b, r parameters, respectively.

We enforce that the total sums of all b, r attributes of the stored payment certificates from v, never exceed the initial given b, r parameters in the `give_credit` instruction. Whenever u receives payment certificate PC from v, the values of $PC.b, PC.r$ are checked to be lower than $D_u^{FSP}[peer = v].b, D_u^{FSP}[peer = v].r$ respectively (Algorithm 11, line 5), and if PC is stored, the values of $D_u^{FSP}[peer = v].b, D_u^{FSP}[peer = v].r$ are reduced by $PC.b, PC.r$, respectively. Also, $D_u^{FSP}[peer = v].b, D_u^{FSP}[peer = v].r$ may increase only when PC is timed out, and they are increased by $PC.b, PC.r$, respectively (Algorithm 18. lines 10,11).

According to Sub-Claim 3, u will never send redemption against payment certificate PC unless the rate meets the LB limitation with $PC.b, PC.r$ as parameters.

The only way for an honest peer to send a message, is using the $sign\_send()$ function (Algorithm 6). The LB limitation is done in Algorithm 6, at line 2, using the function $verify\_LB(DB, ind)$ (Algorithm 5). The verification is done only for non-redemption messages, since the redemption messages rates are pre-allocated, and the verification is done against the payment certificate, as we mentioned above.

The function $verify\_LB(DB, ind)$ verifies if sending a single message at the time of calling it, is valid with respect to the LB model with the parameters $DB[ind].b, DB[ind].r$. It uses the attribute $DB[ind].x$ which represent the size of the bucket of the pending messages on this link, and updates $DB[ind].x$ according to the attribute $DB[ind].t$, which is the last update time.

The b, r parameters of the LB verification are the remaining $D_u^{FSP}[peer = v].b, D_u^{FSP}[peer = v].r$ attributes, after the allocation of the rates of the redemptions. Also, the values of $D_u^{FSP}[peer = v].t, D_u^{FSP}[peer = v].x$ are initialized to the initialization time, and to zero, respectively (Algorithm 8 line 2), and $D_u^{FSP}[peer = v].x$ is updated upon each sending of a message of any type (Algorithm 6. line 11).
■

Proof of Claim 3

Claim 3 *For every honest peer u, at every step of running the game,*

$$Credit(u, v) \geq -D_u^{FSP}[peer = v].balance$$

Proof. The balance that honest party u saves to its peer in $D_u^{FSP}[peer = v].balance$ is the amount that u owes to v, that is, when this value is negative v owes to u money. Obviously, the $Credit(u, v)$ is always non-negative, by definition. So we need to show that even if the balance is negative, it never get lower than $-Credit(u, v)$.

According to Claim 1, $D_u^{FSP}[peer = v].balance$ is initialized to zero, and decreases only when external payment is certified and on sending redemption. On external payment certification (Algorithm 15) the balance never gets lower than zero, according to the condition at line 1, so it still greater than any negative value. So, the only possible way to the balance to be decreased to a negative value, is by sending redemption to v, when the balance is decreased by the net amount of the redemption.

An honest peer u saves in $D_u^{FSP}[peer = v].credit$ the remaining credit for its peer v. This attribute is initialized to $Credit(u, v)$ in Algorithm 8 at line 2. This attribute is checked and updated, upon every received payment certificate from v, that the net amount of the maximal amount that can be redeemed using the received payment certificate is lower or equal to $D_u^{FSP}[peer = v].credit$, and reduced by this value (Algorithm 11 lines 4 and 7). Also, when PC is timed out, $D_u^{FSP}[peer = v].credit$ is updated to be increased by the same value (Algorithm 18. line 9).

According to Sub-Claim 4, all the redemptions against some payment certificate are bounded by the maximal net amount of it, and the sum of all these amounts among all payment certificates from some peer, is bounded by this peer's initial credit. ■

Proof of Claim 4

Claim 4 *If an honest peer u received a payment order PO from its peer v at time t, and PO was acceptable, at that time t, with respect to some payment certificate PC that u issued for v, then u should output*

$$(\texttt{valid\_redemption}, v, PO.id, PC.net(PO.amt))$$

immediately.

Proof. An honest peer evaluates the receiving of payment order in Algorithm 13, and should output the required message $(\texttt{valid\_redemption}, v, PO.id, PC.net(PO.amt))$ if the condition at line 1 is satisfied.

This condition examines the return value of the function $is\_acceptable(PO, PC.id, t)$, where t is the time of running. According to Sub-Claim 6, $is\_acceptable(PO, PC.id, t)$ returns true only if PO is acceptable, at time t, with respect to PC. So, if it is given that PO is acceptable, at that time t, with respect PC, the required message should be output immediately as required.

■

Proof of Claim 5

Claim 5 *Let x be a record in database D_u^{PC} of an honest peer u, and let v denote $x.by$. If v is honest, there is a record y in D_v^{PC} s.t. the tuple:*

$$(x.pc\_id, x.by, x.for, x.path, x.expire, x.trt, x.max, x.net, x.b, x.r, x.v, x.sig)$$

is exactly the same as the tuple:

$$(y.pc\_id, y.by, y.for, y.path, y.expire, y.trt, y.max, y.net, y.b, y.r, y.v, y.sig)$$

Proof. The only possible place to insert a record to D_u^{PC}, where the *by* attribute is not u, is in Algorithm 11. at line 6. To evaluate this algorithm, u must receive an input message $(\texttt{pc\_issued}, pc)$ from v (Algorithm 2, line 11). Moreover, the input pc is passed to Algorithm 11 and when the record is inserted to D_u^{PC}, the attributes

$$(pc\_id, by, path, expire, trt, max, net, b, r, v, sig)$$

of the record are set to the input pc.

Also, if message was sent from honest v, the only possible place to send such message is in Algorithm 17 at lines 7,25. To reach one these lines, v must insert a record which its attributes

$$(pc\_id, by, path, expire, trt, max, net, b, r, v, sig)$$

are set to the pc that it sends to u (lines 6, 24 respectively).

■

Proof of Claim 6

Claim 6 *Let u be an honest party, and let x be a record in D_u^{PC}, where $x.by$ is honest too. Also, let v denote $x.by$, and let po be some payment order.*

For any time t_1, if u runs the function

$$is\_acceptable(po, x.id, t_1 + Delay(u, v))$$

at time t_1 and it returns true, if v runs

$$is\_acceptable(po, x.id, t_2)$$

at t_2 s.t. $t_1 \le t_2 \le t_1 + Delay(u, v)$, it will return true as well.

Proof. The implementation of the function $is\_acceptable()$ is in Algorithm 12. and it returns true only if all the conditions at lines 2 - 9 are satisfied. We list the conditions, and show for each that if it is satisfied in u, it will be satisfied in v as well.

1. Line 2, to satisfy this condition there must be a record in D_v^{PC} with id attribute equals to pc_id.
   According to Claim 5. if there is a x record in D_u^{PC} where $x.by = v \wedge x.id = pc\_id$, there must be a record y in D_v^{PC} with the same PC attributes. Specifically, $y.id = x.id = pc\_id$.
2. Line 3, this condition only depends on the parameter po, which is the same when running in u. So, if it was satisfied in u it will be satisfied in v.
3. Line 4, this condition checks that there is no record in D_v^{PO} with id attribute that equals to $po.id$ from the same issuer.
   The payment order id attribute is unique per issuer (Algorithm 19).
   An honest peer must run the function $is\_acceptable()$ with a payment order, before sending a redemption with it (Algorithm 13. line 1 , Algorithm 20. line 3). Insertion of a payment order record to D^{PO} can take place only after successful check of $is\_acceptable()$ with this payment order (Algorithm 13. line 2 , Algorithm 20. line 4). For $is\_acceptable()$ to return true, our condition (line 4) must be satisfied, i.e. there is no such record with the same $po.id$ and $po.issuer$ in the machine database.
   So, every payment order can pass $is\_acceptable()$ only one time per machine. Also, due to the condition at line 9 (specific redemption path for PO) that is satisfied in u, a payment order can pass $is\_acceptable(PO, pc\_id, time)$ in v, only if it comes from u. So, if when u runs $is\_acceptable$, there was no record in D_u^{PO} with id and $issuer$ attributes that equals to $po.id$ and $po.issuer$, respectively, then when v will run it there will be no record in D_v^{PO} with id and $issuer$ attributes that equals to $po.id$ and $po.issuer$, respectively.
4. Line 5, $D_v^{PC}[pc\_id].net(po\_amt) \leq D_v^{PC}[pc\_id].avl$.
   The left side of the inequality should be the same when run by u, since $D_v^{PC}[pc\_id].net = D_u^{PC}[pc\_id].net$, according to Claim 5 as we showed above. We will show that $D_v^{PC}[pc\_id].avl \leq D_u^{PC}[pc\_id].avl$ by induction. The attribute avl in both records is initialized to the attribute max when it is inserted to D^{PC} (Algorithm 17 lines 6,24, and Algorithm 11, line 6).
   The attribute $D_u^{PC}[pc\_id].avl$ is reduced only in two places
   (a) Algorithm 17. at line 16, when u issue payment certificate that is derived from the payment certificate with the identifier pc_id. The subtraction is by the max attribute of the derived payment certificate.
   (b) Algorithm 20. at line 6, when u sends redemption of PO that was received from the upper layer and passed $is\_acceptable(PO, pc\_id, time)$, to v.
   The attribute $D_v^{PC}[pc\_id].avl$ is reduced by the net amount of payment order PO that passes the check of $is\_acceptable(PO, pc\_id, time)$ (Algorithm 13, line 3). As we showed before, such PO can be received only from u.
   So, if v reduced $D_v^{PC}[pc\_id].avl$ by the net amount of some PO from u, u must have reduced equal or greater amount from $D_u^{PC}[pc\_id].avl$, because

the net amount of the payment orders that came from a derived payment certificate are bound by the *max* attribute of the derived payment certificate, and the net amount of a payment certificate that came from upper layer is equal in both sides.

5. Line 6, the validation of the redemption rate.
   The validation of the LB, using the function $verify\_LB()$, is done with the same LB parameters, since according to Claim 5, $D_v^{PC}[pc\_id].b = D_u^{PC}[pc\_id].b$ and $D_v^{PC}[pc\_id].r = D_u^{PC}[pc\_id].r$.
   The attribute x is updated in v upon each redemption of payment order from u (Algorithm 13. line 4), and updated in u as well (Algorithm 20. line 7, and Algorithm 13. line 12).
6. Lines 7 and 8, check that *po* is up to date, and can be redeemed before the expiration time, according to the *trt* attribute in the payment certificate.
   Of course, the right sides of the inequalities are the same when run by u and by v, according to Claim 5. The only difference is the *time* parameter. Recall that u runs *is_acceptable* with *time* parameter equals to $t_1 + Delay(u, v)$ while v runs with *time* parameter t_2. Since $t_2 \leq t_1 + Delay(u, v)$, according to Claim 2, the left sides of the inequalities will be equal or lower when run by v than they were when run by u, and thus, the conditions will be satisfied.
7. Line 9. This condition depends on *po*, which is the same when run by v as when run by u, and on $D_v^{PC}[pc\_id].path$ that according to Claim 5, $D_v^{PC}[pc\_id].path = D_u^{PC}[pc\_id].path$.
   So, if is satisfied for u, it is satisfied for v as well.

So, we showed that for each condition in *is_acceptable*, if it is satisfied in u, it will be satisfied in v as well. ■

Proof of Claim 7

Claim 7 *Let pc_derived be a derived payment certificate that an honest peer u issued based on payment certificate pc_base from p. Every payment order that would be acceptable with respect to pc_derived at time t, will be acceptable with respect to pc_base at any time* $t_2 | t \leq t_2 \leq t + Delay(u, p)$.

Proof. The only place that an honest party can issue *pc_derived* is in Algorithm 17. at line 25. To reach this line, the condition that ends at line 15 must be satisfied. According to Sub-Claim 5, this condition is satisfied only if every acceptable payment order with respect to *pc_derived* at some time t, is also acceptable with respect to *pc_base* at any time $t_2 | t \leq t_2 \leq t + Delay(u, p)$, where p is the issuer of *pc_base*.
■

8 Liveness

In Section 7 we proved safety property of the Inter-FSP Funds Transfer Protocol. In this section we prove the liveness property, i.e. that the protocol reacts upon instructions without throwing exceptions. Actually, in most of the instructions of the protocol, the protocol throws exceptions only for trivially invalid input. However, in the instructions `issue_pc` and `po_received`, it is not trivial to see that exceptions do not occur. We need to show that for these instructions, the Inter-FSP Funds Transfer Protocol throws exceptions only for invalid instructions.

Definition 10 *We define the expected behavior of $\mathcal{M}^{FSP}$ upon* $(\texttt{issue\_pc}, PC\_attributes)$ *instruction, as follows: The machine should sign a payment certificate PC that consists of the given attributes, as defined in Definition 5, and send* $(\texttt{pc\_issued}, PC)$ *message to* $PC.for$*. This should be done under the restriction that redemption of any payment order that would be acceptable with respect to PC, at any time, will not cause money loss as defined in Definition 4.*

Definition 11 *We define the expected behavior of $\mathcal{M}^{FSP}$ upon* $(\texttt{po\_received}, PO, PC_{id})$ *instruction, as follows: The machine should send* `redeem` *message with the given payment order PO to the peer that issued the given payment certificate. This should be done under the restriction that the given payment certificate was received from other peer, and that PO is acceptable with respect to it on the time of receiving the redemption.*

Definition 12 *We say that $\mathcal{M}^{FSP}$ has the liveness property if it behaves as expected upon the instructions* `issue_pc` *and* `po_received`*.*

Theorem 3. *The Inter-FSP Funds Transfer Protocol has the liveness property.*

Proof. We show for each instruction, that the Inter-FSP Funds Transfer Protocol behaves as defined in Definition 10 and Definition 11:

1. The instruction $(\texttt{issue\_pc}, PC\_attributes)$ is evaluated in Algorithm 17. In case of prime PC, the required payment certificate is sent at line 7. In case of derived PC, the required payment certificate is sent at line 25, only if the condition that ends at line 15 is satisfied. According to Sub-Claim 5, this condition is satisfied if every acceptable PO with respect to the derived PC at some time t, is acceptable with respect to the base PC at any time $t_2 | t \leq t_2 \leq t + Delay(u, p)$, where p is the issuer of the base PC. Of course, if the machine receives PO that is acceptable with respect to its derived PC, and not acceptable with respect to the base PC on the time of receiving the redemption, it will lose the money according to Definition 4. So, the condition that ends at line 15 validates exactly the restriction of Definition 10.
2. The instruction $(\texttt{po\_received}, PO, PC_{id})$ is evaluated in Algorithm 20. The `redeem` message is sent as required at line 8, only if the condition at line 3 is satisfied. This condition checks that the given PC was received by machine u from other peer p, and that the function $is\_acceptable(PO, PC_{id}, now() + Delay(u, p))$ returns true. According to Claim 6, if $is\_acceptable(PO, PC_{id}, now() + Delay(u, p))$ returns true in u, $is\_acceptable(PO, PC_{id}, t_2)$ must return true

in p, for any time $t_2 | t \leq t_2 \leq t + Delay(u, p)$. According to Sub-Claim 6, $is\_acceptable(PO, PC_{id}, t_2)$ returns true in p only if PO is acceptable with respect to PC at time t_2. So, the condition at line 3 validates exactly the restriction of Definition 11.

■

9 Conclusions

- We present the Inter-FSP Funds Transfer Protocol. The protocol ensures reliability by limiting the service rate for each peer.
- We designed the protocol with conservative risk management, that ensures the party will fulfill its commitments even in the worst case. Note that, real systems may rather take more risks to maximize the revenues, and a system with more freedom degree should be implemented based on our work. For instance, one can limit the risk by letting each party to estimate the probability of loss in addition to the credit it gives to its peer, and require that the probability of bankruptcy is less than some parameter.
- The proposed protocol is handling the relationships and transaction between peers. Finding a path of peers that trust and work with each other, is out of this protocol scope. The difference between these two task is similar to the difference between the IP protocol and routing in networking. A future work can be done on the routing area, i.e. finding best path on a given FSP network.
- Adding an evidences layer to our protocol could provide acceptable evidences for disputing, outside of the protocol, in the cases that are considered as loss. These evidences should include non-repudiated delivery of messages and signed certificates that the peers give to each other, for example see the work of Herzberg et al. [10].

Bibliography

[1] Swift instructed to disconnect sanctioned iranian banks following eu council decision, March 2012. URL `http://www.swift.com/news/press_releases/SWIFT_disconnect_Iranian_banks`.

[2] M. Backes and M. Duermuth. A cryptographically sound Dolev-Yao style security proof of an electronic payment system. 2005. ISSN 1063-6900.

[3] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *Proceedings of the 10th ACM conference on Computer and Communications Security*, pages 220–230. ACM, 2003. ISBN 1581137389.

[4] M. Bellare, J.A. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, and M. Waidner. ikp: a family of secure electronic payment protocols. In *Proceedings of the 1st conference on USENIX Workshop on Electronic Commerce-Volume 1*, pages 7–7. USENIX Association, 1995.

[5] Jan Camenisch, Ueli Maurer, and Markus Stadler. Digital payment systems with passive anonymity-revoking trustees. *Journal of Computer Security*, 5(1):69–89, 1997.

[6] D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Proceedings of Crypto*, volume 82, pages 199–203, 1983.

[7] Y. Chen, R. Sion, and B. Carbunar. Xpay: Practical anonymous payments for tor routing and other networked services. In *Proceedings of the 8th ACM workshop on Privacy in the electronic society*, pages 41–50. ACM, 2009.

[8] D. Dolev and A. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, 1983.

[9] A. Herzberg, E. Shai, and I. Zisser. Decentralized electronic certified payment, 2009. US Patent 7,546,275.

[10] Amir Herzberg and Igal Yoffe. The delivery and evidences layer. *IACR Cryptology ePrint Archive*, 2007:139, 2007.

[11] Amir Herzberg and Igal Yoffe. The layered games framework for specifications and analysis of security protocols. *International Journal of Applied Cryptography*, 1(2):144–159, 2008.

[12] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*, volume 2050. Springer, 2001.

[13] Harry Leinonen and Kimmo Soramäki. Simulating interbank payment and securities settlement mechanisms with the bof-pss2 simulator. Technical report, Bank of Finland, 2003.

[14] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2009.

[15] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*, pages 184–200. IEEE, 2001.

[16] M. Schmees. Distributed digital commerce. In *Proceedings of the 5th international conference on Electronic commerce*, pages 131–137. ACM, 2003.

[17] D. Stiliadis and A. Varma. Latency-rate servers: a general model for analysis of traffic scheduling algorithms. *IEEE/ACM Transactions on Networking (ToN)*, 6 (5):611–624, 1998.

[18] L. Xiong and L. Liu. A reputation-based trust model for peer-to-peer ecommerce communities. 2003.

[19] B. Yang and H. Garcia-Molina. Ppay: micropayments for peer-to-peer systems. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 300–310. ACM, 2003.

[20] Dan Zhu. Security control in inter-bank fund transfer. *Journal of Electronic Commerce Research*, 3(1):15–22, 2002.