Inter-FSP Funds Transfer Protocol
Technical Report

Shay Nachmani and Amir Herzberg

Computer Science Department, Bar Ilan University

Table of Contents

Abstract. The present work introduces the first *decentralized secure funds transfer protocol* with multiple participants. The protocol guarantees that a participant only loses money if a trusted peer happens to be corrupt. Furthermore, the loss is limited to the amount of credit given to that partner. The protocol supports expiration times for payment orders, and takes into consideration actual network queuing delays. To achieve our goals, we used several models and techniques from the Quality of Service area, to handle delays and avoid the expiration of payment orders. We provide rigorous proofs to the security requirements of the protocol.

1 Introduction

The present work discusses electronic funds transfer, a basic and critical financial operation. We use the term *funds transfer* to denote money transfers between FSPs [1], as opposed to regular *payments* that are usually made for certain goods. Consider a client that wishes to buy certain goods from vendor A on the Internet. The client is unable to pay A directly, as no payment mechanism exists between them. He must therefore provide a payment commitment from FSP B with whom A works. This funds transfer operation is carried out in several stages, which are illustrated in Fig. 1. Initially, FSP B issues a payment certificate (PC) for its partner A. A PC is an electronic signed certification stating that B commits to transfer funds against payment orders (PO), subject to certain restrictions. A PO is an electronic signed message that orders the payment of money. The restrictions in the PC may limit the total amount of the POs, set a deadline for the commitment, define a commission rate, etc. The PO itself may also have an expiration date. FSP B may issue a PC at the request of its client, stating that the latter has deposited money in B and wishes to transact with A. The client can pay A with the PO issued for him by B. Once A receives the PO, a redemption request can be sent to B, which should transfer the funds it has committed to. Our protocol covers the stages of issuing the PC and redeeming the PO. These stages are indicated in Fig. 1 by continuous lines. The actual payment and the transfer of the goods are not dealt with here, and are indicated in Fig. 1 by broken lines. Nevertheless, our protocol is informed about these operations, and reacts accordingly. This operation is somewhat complex. To begin with,

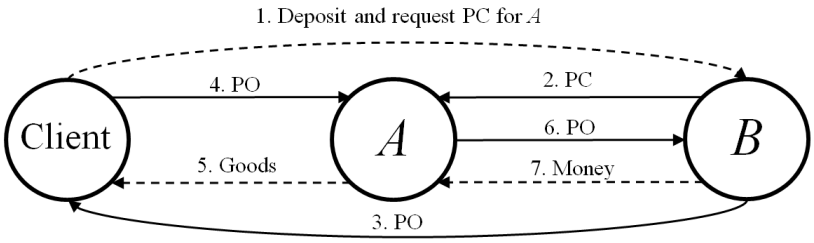

Fig. 1. In an electronic funds transfer operation, B commits in a PC to pay money to A against a PO, and the client pays A with B's PO

trust must exist between two parties that wish to transact with each other. There is no guarantee that the committed party will indeed pay the money, whether due to corruption or to bankruptcy. In order to cover payment commitments, funds must be reserved. This implies that the commitment must have a defined expiration date, to enable releasing reserved funds. It follows that the operation depends to a large extent on communication. If a party fails to send redemption request, or if the network is overloaded and the deadline is passes before the request is received, money will be lost. Therefore, the end-to-end

[1] Financial Service Providers, financial institutes, e.g. banks, insurance companies and investment funds, that have reciprocal business relations with each other and handle accounts for their clients.

communication delay should be taken into consideration. Of course, a queuing delay depends on network loads; to determine their bounds QoS techniques must be used. These problems can be dealt with rather easily if all the involved parties have a trusted third party (TTP) that will make the payment or resolve communication conflicts that come up between them. Most contemporary systems and research solutions take on this centralized approach. Centralized designs are simple to outline, launch and operate; however, they also have significant disadvantages, which motivated us to focus on a decentralized design in this work. Below are several of these disadvantages:

Lack of competition: Obviously, a TTP receives a commission from the parties who wish to transfer funds between them. Very few TTPs have the trust of both sides. As a result, there is little competition, and the costs are relatively high.

Reduced interoperability: In the centralized model, it may happen that certain entities do not have a common TTP, and are therefore unable to carry out any financial operations with each other.

Bottleneck: Centralization faces the TTP with significant loads and overheads, as all the traffic passes through it.

Single point of failure (and 'legal attack'): A technical failure in the TTP is liable to disrupt the entire network, and make it vulnerable to denial-of-service attack. An example illustrating this point is the instruction given to SWIFT by the European Council to stop providing services to Iranian banks subject to European Union sanctions [1], making it impossible to trade with Iran.

We introduce here the first provably secure decentralized funds transfer protocol. Our solution allows transactions on the Internet through multiple FSPs, each interacting with and trusted by its own partners only, without having to rely on one common trusted third party. Decentralization should predictably increase competition between the payment providers, reduce commissions and costs, and improve availability and connectivity. Decentralization also facilitates the participation of smaller players, disproving the assumption that trading systems are all highly reputable large institutes. It should be taken into account that the smaller parties may not behave properly, and that the decentralized model creates new challenges, where several parties might cooperate against another party.

1.1 Network Limitation Aspects

The quality of service in networks has been widely researched; however, the influence of network delays on electronic trade has not been explored as yet. Making certain assumptions on the network, such as a minimum transmission rate, we are able to guarantee that if the involved party behaves according to the proposed protocol, it will not lose money due to excessive network delays.

In order to appreciate the communication challenges that exist in a decentralized system, consider the case illustrated in Fig 2. Four FSPs are involved: A, A', B, and C. First, consider a simple case: a PO issued by C, that A sends to B for redemption. Suppose the PO expires at time t; clearly, B must receive it a bit earlier, say before $t - \delta$, to make sure B can send the PO so it will be received by C in time. Otherwise, a payment order may be sent too late, causing B (or A) to lose funds. Note that if such loss is allowed, it may also be abused, e.g., if A loses, then B may fake a network failure.

Next, note that determining this delay δ is not trivial, as it may depend on other transmissions, e.g., concurrent redemption requests that A' may be sending to B (say, of another payment order issued by C).

Who will be losing money if A and A' send redemption requests with POs to B, at higher rates than B is able to handle and send to C? Obviously, C will not pay the funds; must B still pay A and A', should they somehow split the loss?

If it were decided that B should be the one to bear the loss, the other parties A, A' and C might collude to make B lose. However, if it were decided that A or A' should be the losers, B and C might collude to make A or A' lose. Note that if the participants (e.g. A,A',B, or C) are reputable entities such as banks, they would obviously not abuse the system this way. However, the protocol is designed to be safe even for less trustworthy participants (i.e. smaller and less reputable entities). It is possible to avoid these losses using QoS techniques. Our solution addresses for the first time the delays caused by queuing and congestion, namely, it takes into consideration the fact that the network's capacity is limited. This point is crucial when the discussion concerns delivering payment orders and commitments that have expiration dates.

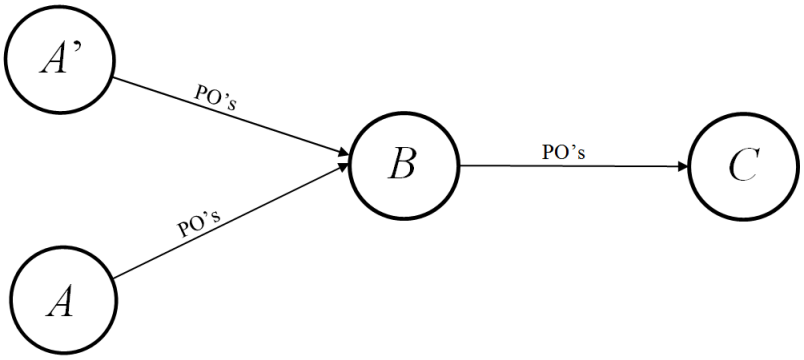

Fig. 2. Funds Transfer with Four FSPs

The protocol limits the rate at which redemption requests are sent. This rate restriction is formulated using the 'leaky bucket model',[12] (Section 4.2), which allows for setting a maximum rate and a maximum pending request bucket to limit the transmission rate. By this method, an honest peer is able to set limits in its PC to the received redemption requests rate in accordance with the rate restriction of the peer that comes next on the redemption path.

Additionally, by limiting the overall transmission rates between peers using the leaky bucket model, the protocol limits maximum delay. Hence, a peer is able to calculate the extra time required to send a redemption request before it expires. To this end, we assume that the underlying layer guarantees the delivery of messages at a minimum rate with certain latency. This assumption is based on a popular model known as the 'latency-rate server' model [12], and is discussed in the communication model (Section 3.3).

1.2 The Trust Model

When financial institutes such as banks do business with each other, they decide the credit amount that would be given to each partner. This credit is the maximum amount to be risked by trusting a certain partner, taking into account the peer's reputation, and the probability that this peer or its employees would go bankrupt or commit fraud. The proposed protocol provides an automated risk limiting mechanism, where the risks are determined externally by the upper layer. In the protocol, each party sets its credit for each peer, and the protocol guarantees that the possible loss to be caused by a peer does not surpass this credit. Moreover, the protocol guarantees that a party does not lose money unless its peer is corrupt. That is, risk limiting is *conservative*, because honest participants are required to limit the total risk for all their peers so that it does not reach an amount that could lead to bankruptcy. Of course, if all the parties follow the protocol correctly, no loss should ever occur. To understand the trust model, consider the following scenario, illustrated in Fig 3. FSP A gives FSP B a 1M$ credit, namely, A allows B to owe it 1M$. B can use this credit to issue a PC for A, i.e. commit to pay A against its own POs. A honors these POs, which may be received from its clients, to the amount of up to 1M$ until B pays it the money. Now consider that B credits FSP C with 2M$, and C issues a PC for B. In this situation, B can issue a PC for A committing to redeem POs of up to 2M$ from C as well. This way, funds can transfer from C to A, B pays A against the POs of C, and is later paid by C. As already mentioned, all the PC's have deadlines and limited amounts. Therefore, the protocol must limit the commitment B may make in its PC, based on C's PC, to avoid loss. As already mentioned, there are also network limitations to consider, and B should limit the rate of the POs sent from A, so that they can be successfully redeemed by C. Moreover, if B wishes to issue a PC for another peer A', committing to redeem C's POs (see Fig. 2), the credit for C should be divided between A and A' in the PCs; the redemption rate and the amount of C's PC should also be divided between them. Obviously, fraud is a prevalent concern in online transactions; double spending, forgery, and repudiation must be avoided. Therefore, we bring rigorous cryptographic proof that the proposed payment system is secure. We also introduce a general model for validating a protocol's security requirements. In spite of the available

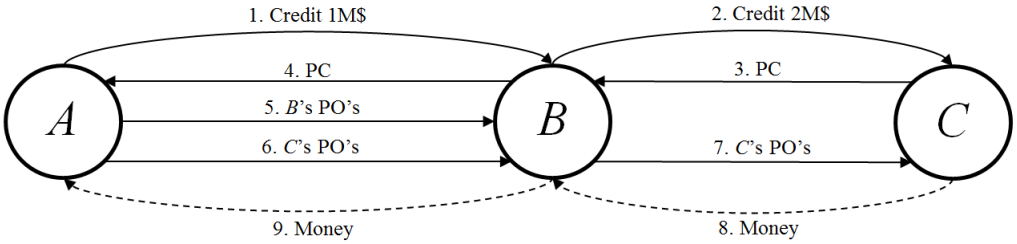

Fig. 3. Protocol scenario with three FSPs

variety of automatic proof systems, we have not come across a system that is able to model service rate limitation, and prove the network's delay related requirements.

1.3 Our Contribution

We propose a decentralized funds transfer protocol, allowing multiple FSPs.

- The protocol guarantees that a loss can only be caused to the involved parties by a corrupt peer, and the amount of this loss does not surpass the credit given to this peer.
- We address the network dependency aspect, taking into account the limitations of the participants' communication capabilities, and the effect of the communication delay on financial commitments.
- To validate the requirements, we use a well-defined generic model that takes into account the assumed communication model of the lower layer. This model can be used for other protocols as well.
- The protocol supports FSP commissions over the redemption path, for the service of transferring the funds.

Organization In Section 2 we review other works related to ours. In Section 3 we describe the model of the system, the assumptions related to the upper and lower layers and the services they provide. 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 extensively 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 for the funds transfer we discuss, but does not include a security analysis and does not deal with network delay aspects.

Several research works are available regarding interbank funds transfer. In [13], Leinonen et al. describe a payment and settlement system simulator they have developed, which can be used to construct simulation models of payment systems. Zho [20] examined the operation flow of the electronic funds transfer process and its security control mechanism. None of these works handle the risk limiting and network issues we deal with in our work.

2.2 Electronic Payment Systems

Much work has been carried out on anonymity [5, 6, 7]. We do not deal with this issue at this stage, but consider integrating it into a future work.

Following is a review of the research carried out to date on open decentralized payment systems. Schmees [16] describes the benefits gained by giving up a centralized client- server payment system and moving toward distributed electronic payment using peer-to-peer networks. However, he does not propose a mechanism for implementing this model.

Yang and Garcia-Molina describe a micropayment system protocol called PPay [19], built upon a peer-to-peer network. They introduce the concept of floating, a self-managed currency concept that allows digital currency to float from one node to another without involvement of a TTP. Although PPay security is based on digital signatures, fraud cannot be prevented; instead, a mechanism of punishment and risk management is proposed that makes fraud unprofitable.

Xiong et al. developed PeerTrust [18], a peer-to-peer trust model that enables quantifying and comparing the trustworthiness of peers by a transaction-based feedback system. They built a general trust metric that effectively measures and captures the trustworthiness of peers, and also addresses the issue of fake or misleading feedbacks, minimizing the risk involved in e-commerce transactions. However, the model does not provide any protection against certain outcomes 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, adding them to the end of the coin. A payee is able verify the signatures and thus verify the chain of ownership. To prevent double-spending, a peer-to-peer network is proposed, using proof-of-work to record the public history of transactions. Fraud becomes computationally impractical for an attacker if honest nodes control most of the CPU power.

Notably, none of the above works contains signed commitments between peers, with deadlines. Thus, the challenges are extremely different from those of our work. What is more, none of them deals with the influence of network delay on the payments.

2.3 Formal Security Proofs

There are two approaches to conducting a security proof:

The cryptographic approach Usually proved by reduction to one of the underlying cryptographic primitives. This approach provides rigorous proofs but they are not automated.

The automated formal-methods approach , based on the Dolev-Yao model [8]. The model represents cryptography as term algebras, and simplifies security proofs for large protocols. In [3] Backes proved that the real cryptographic library with its much more sophisticated adversary is as secure as the ideal cryptographic library. Therefore, a protocol that is proved based on the deterministic Dolev-Yao-like ideal library can be safely implemented on the real cryptographic library. For example, in [2] Backes used this model to prove that a 3KP electronic payment system is secure [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 the security analysis of our protocol.

3 General Security Model

In this section we describe the protocol's execution model, and introduce the general model of validation for any set of requirements under any set of communication assumptions. This model may be used for various protocols and other purposes, with different requirements and different communication models. The model is based on the work of Herzberg and Yoffe [11].

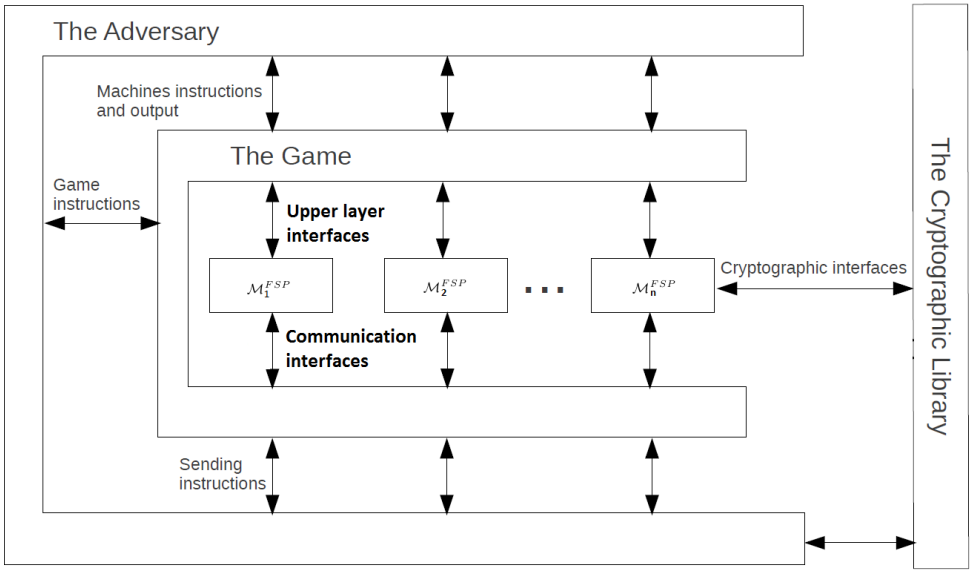

Fig. 4. Overview of the Game

3.1 Protocol Execution Model

Each party is represented by interacting machine $\mathcal{M}$ which reacts to 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 their complexity 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, whose output is also polynomial in the initial input. This brief definition is taken from [3]. The machine is controlled by its administrator via internal input commands. The machine passes relevant information to the administrator from received external input.

3.2 Requirements Model

We define a game in which the adversary sets honest parties and corrupt parties. The honest parties follow the protocol, with the adversary as administrator. They also have reliable communication with each other. The corrupt parties are fully controlled by the adversary. The honest parties and the adversary use the cryptographic library directly. The adversary controls all the communications in the game. All the machines are given a global time parameter; the time is received from the adversary, and is verified to be non-decreasing. For an overview of the game see Fig 4.. The game receives the adversary algorithm as input, and executes its instructions step by step. These instructions might involve sending messages to honest parties, or giving the administrator instructions for them. The adversary is a machine $\mathcal{A}$ that has the following interfaces:

- (`init_request`, 1^k) receives 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 $\mathcal{M}$ initialized by the game.
- (`next_state`, adv_state) receives the current $\mathcal{A}$ state and returns the tuple ($instruction$, id, $parameter\_list$, $time$, adv_s where $instruction$ may be the name of an instruction to $\mathcal{M}$, or an instruction given tothe game to end the game or to send a message to one of the parties. id is the identifier of the $\mathcal{M}$ that should receive this instruction or message; $parameter\_list$ is the 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) receives the $\mathcal{M}$ output, sends a request with the current state, and returns the next state of $\mathcal{A}$.

The game ensures that the time is strictly progressing, and no more than one operation can occur at a certain point in time. The game provides both the parties and the adversary with a cryptogaphic library. This library receives the security parameter, and has interfaces for generating randomize id and key pairs. It also has interfaces for signing and verifying messages which could be broken with negligible probability with respect to the security parameter. An example for a library of this kind is found in [3]. The game stores a communication log and a protocol log, where every instruction given to an honest party and every ensuing output is logged. When the adversary gives an instruction to end the game, the game returns true if the adversary has won, i.e., the protocol log is valid (no cheating), or the communication log is invalid (the adversary did not preserve the assumptions about the communication layer). That is to say, 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 for specific protocols and communication models. The game receives these functions as parameters. The implementations that follow our requirements and communication model are described in the next section. For the game pseudo code, see Algorithm 1 below.

Definition 1 *We state 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 a simple model in which the machines communicate over a network whose peers are able to handle each other's requests with some delay, and then with a fixed rate of service. This model is called a latency rate server, [17] and 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. The LR server model is widely used in the literature, and many well-known scheduling algorithms, such as TDM, Fair Queueing, VirtualClock, and Weighted Round Robin, have been proved to belong to the class of LR-servers. The behavior of an LR server is determined by two parameters L and R as follows: After the initial latency L, the scheduler allocates to the session a minimum service rate R. That is to say, the model guarantees delivering $(t - L) * R$ bits in intervals the length of which is t. For a graphic description see Fig. 5. We assume that all the links have the same parameters, to avoid needless complexity. We also assume reliable FIFO communication between the 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 following:

1. Communication service rate according to the 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.

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

$ProtocolLog = ""$;
$CommLog = ""$;
$CurrentTime = 0$;
$H = \phi$;
$(adv\_state, communication\_parameters, n) = \mathcal{A}(\text{init\_request}, 1^k)$;
for $i = 1 \rightarrow n$ **do**
    $(state, id) = \mathcal{M}^{FSP}(\text{init\_request}, 1^k, communication\_parameters)$;
    $States[id] = state$;
    $H = H \cup \{id\}$;
`// The first line of the log is for the parameters`
$ProtocolLog+ = (\text{Parameters}, communication\_parameters, n, H)$;
$CommLog+ = (\text{Parameters}, communication\_parameters)$;
while True **do**
    $(instruction, id, parameter\_list, time, adv\_state) = \mathcal{A}(\text{next\_state}, adv\_state)$;
    if $time \leq CurrentTime$ **then**
        return False;
    $CurrentTime = time$;
    if $instruction = \text{finish\_game}$ **then**
        $ProtocolLog+ = (\text{finish\_game}, time)$;
        return `Valid_CommLog(`$CommLog$`)` $\wedge$`!Valid_ProtocolLog(`$ProtocolLog$`)`
    if $instruction = \text{send}$ **then**
        $(u, msg) = parameters\_list$;
        `// sending_request is a (destination, message) list`
        $(States[id], sending\_request, out) = \mathcal{M}^{FSP}(\text{received}, States[id], u, msg)$;
        $CommLog+ = (\text{received}, time, id, u, msg)$;
    else
        `// regular instruction to machine`
        $(States[id], sending\_request, out) = \mathcal{M}^{FSP}(instruction, States[id], parameters\_list)$;
        $ProtocolLog+ = (id, time, instruction, parameters\_list)$;
    $adv\_state = \mathcal{A}(\text{protocol\_output}, out, sending\_request, adv\_state)$;
    foreach $(dest, msg) \in sending\_request$ **do**
        $CommLog+ = (\text{send}, time, id, dest, msg)$;
    $ProtocolLog+ = (id, time, out)$;

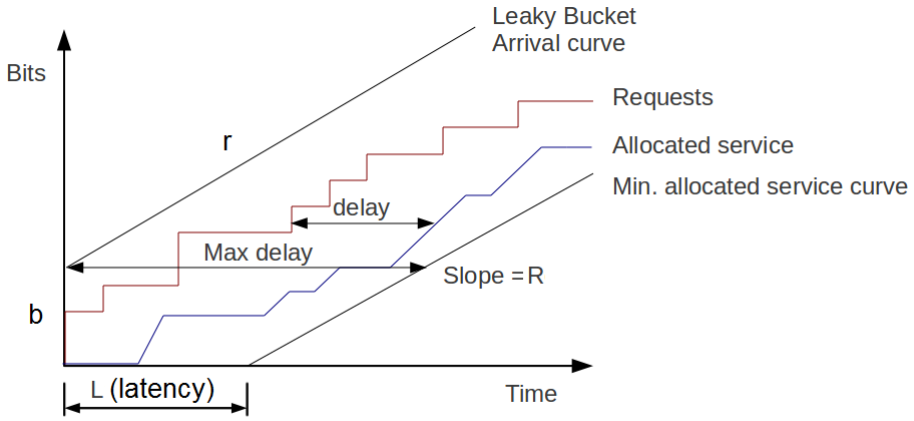

Fig. 5. Latency Rate Server

Algorithm 2: LR_FIFO_Valid_CommLog($CommLog$)

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
Let $R_{u,v,t_1,t_2} = |\{(\texttt{received}, u, v, t, msg) \in CommLog | t_1 \leq t \leq t_2\}|$
Let $S_{u,v,t_1,t_2} = |\{(\texttt{send}, u, v, t, msg) \in CommLog | t_1 \leq t \leq t_2\}|$
if $\exists (u, v, t_1, t_2) | \exists t_1 \leq s \leq t_2 \wedge R_{v,u,t_1,t_2} < S_{u,v,t_1,s} - (t_2 - s - L) * R$ **then**
    return False;
// We require FIFO
Let R_MSG_{u,v,t_1} be the concatenation in chronological order of all the messages in $\{msg | (\texttt{received}, u, v, t, msg) \in CommLog \wedge t \leq t_1\}$
Let S_MSG_{u,v,t_1} be the concatenation in chronological order of all the messages in $\{msg | (\texttt{send}, u, v, t, msg) \in CommLog \wedge t \leq t_1\}$
if $\exists (u, v, t) | R\_MSG_{v,u,t}$ *is not prefix of* $S\_MSG_{u,v,t}$ **then**
    return False;
return True;

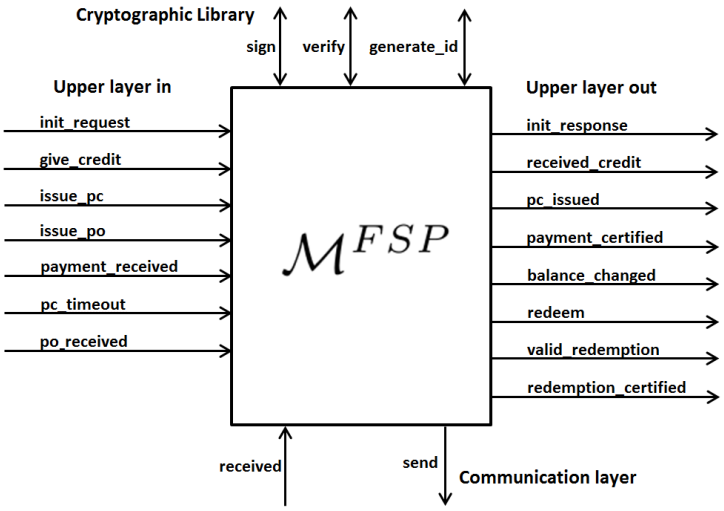

Fig. 6. $\mathcal{M}^{FSP}$ interfaces

4.1 The FSP Machine

The FSP machine is used to manage an FSP entity. It saves the balance for each peer, handles the commitments, and communicates with the other peers. The machines have interfaces to recognize each other via public keys inserted by the administrator. They also have interfaces for allocating credits, sending commitments, redeeming, and certifying external payments. The underlying layer provides interfaces that enable sending and receiving messages. The cryptographic library provides interfaces that enable signing and verifying signatures, and generating identifiers and keys. The protocol is defined by the FSP machine. There is a network of FSPs that set credits for each other; that is, the administrator of FSP_u gives his machine a command to set the credit for FSP_v. Of course, the FSP_u machine must have the public key and address of FSP_v, and should receive them via the set credit interface as the identifier. This means that FSP_u trusts FSP_v and agrees for FSP_v to owe it money up to the credit amount, i.e., FSP_u takes the risk that FSP_v will not pay as committed due to bankruptcy, corruption, or another failure. If the FSP machine does not receive any input regarding the credit set for a partner, the credit for this partner is zero. The machines issue signed commitments - PCs, and checks - POs, and redeem them. The upper layer is also able to notify the machine about external payments, and certification about this payment should be sent to the payer machine. For the sake of intelligibility, we refer to the inputs from the upper layer interfaces as instructions.

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

1. Upper layer interfaces
   - $in(\texttt{init\_request}, security\_parameter, communication\_parameters)$ to initialize an FSP machine with the communication parameters.
   - $out(\texttt{init\_response}, identifier)$ to deliver the identifier given by the underlying layer to the upper layer.
   - $in(\texttt{give\_credit}, peer\_identifier, credit, b, r)$ to give initial credit, and set leaky bucket parameters b, r to limit the rate of communications sent to this peer. The leaky bucket model is discussed in section 4.2 below.
   - $out(\texttt{received\_credit}, peer\_identifier, credit)$ to report a given credit.
   - $in(\texttt{issue\_pc}, PC\_arguments)$ to issue signed commitments - PC (Definition 5).
   - $in(\texttt{pc\_timeout}, PC_{id})$ to notify PC expiration.
   - $out(\texttt{pc\_issued}, payment\_certificate)$ to report that a PC was received (Definition 5).
   - $in(\texttt{payment\_received}, peer\_identifier, amt)$ to notify the payee that an external payment was made.
   - $out(\texttt{payment\_certified}, peer\_identifier, redemption\_id, amt)$ for the payer to report receiving an external payment certification from the payee.
   - $out(\texttt{redeem}, peer\_identifier, redemption\_id, amt)$ to inform the upper layer that a redemption request was sent with a PO (Definition 7).
   - $out(\texttt{valid\_redemption}, peer\_identifier, redemption\_id, amt)$ to report that a valid redemption request was received from another peer.
   - $out(\texttt{redemption\_certified}, peer\_identifier, redemption\_id, amt)$ to report that a redemption certification was received from the peer to whom the redemption request was sent.
   - $in(\texttt{issue\_po}, path, amt, expire)$ to issue a PO.
   - $in(\texttt{po\_received}, PO, PC_{id})$ to redeem a PO received from the upper layer against a PC.
2. Communication interfaces
   - $out(\texttt{send}, peer, message)$
   - $in(\texttt{received}, peer, message)$
3. Cryptographic interfaces
   - $(\texttt{generate\_id}, security\_parameter)$ generates a unique identifier and key pair for signing.
   - $(\texttt{sign}, key, message)$ creates an electronic signature of a message.
   - $(\texttt{verify}, message, key, signature)$ verifies an electronic signature of a message with a key, returns true upon successful verification, and otherwise false.

The interfaces are illustrated in Fig. 6.

Algorithm 3: LB(b,r)

```
1  Initially:
2      bucket = 0 ;
3  Upon request arrival:
4      if bucket + 1 ≤ b then
5          bucket + +;
6          return True;
7      else
8          return False;
9  Every 1/r seconds:
10     if bucket > 0 then
11         bucket − −;
```

4.2 The leaky bucket model

We use the leaky bucket model to limit transmission rate between peers. The model is given rate r, and bucket size b as parameters, and commits to service packets if the arrival rate of the requests is lower than r, with a buffer size of b. A leaky bucket validation pseudo code is provided in Algorithm 3.

Assuming the underlying layer guarantees on a service rate according to LR server model, with given parameters, we are able to limit maximum delay between peers, where the transmission rate is limited by using LB model.

Definition 3 *Let $Delay(u, v)$ denote the upper limit of the delay of the communication sent 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

It is required that an honest party lose money only if its peer is corrupt, and that the loss be limited by the credit set for this peer.

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

Therefore, the protocol must ensure that an honest party always accepts a valid redemption request from its peer. We consider every redemption request sent from an honest party as valid. Also, to limit the loss we limit the possible debt. It is therefore required that the negative balance be limited for each peer. This limitation is set by the upper layer, in the form of the credit set for that peer. In summary, the requirements are the following:

1. Every redemption request sent by an honest party to its honest peer must be accepted.
2. An honest party will never have a peer with a negative balance that surpasses that peer's credit limits.

To validate these requirements we implement the $FSP\_Valid\_ProtocolLog$ function of the game in Algorithm 4. First, lines 2 - 3 set the initial credit 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 calculate the maximum communication delay from u to v. Line 19 in Algorithm 4 validates the first requirement. We verify that every redemption request sent by an honest party to an honest peer is accepted no later than

the maximum communication delay allowed. Of course, if the game ends before the redemption message has arrived, the requirement is no longer justified.

To validate the second requirement, we update the values of the balance of the parties, following redemptions (lines 16, 18) and payments (lines 24, 26). The condition in line 27 ascertains that no negative balance has surpassed the credit limits, as required. The conditions in line 22 are meant to restrict the external payment notifications to zero negative balance, i.e. the payer does not gain a positive balance by an external payment. This way, the adversary is unable to create a negative balance for the payee, which could exceed the credit limit.

5 The Inter-FSP Funds Transfer Protocol

Following is an overview of the Inter-FSP Funds Transfer Protocol, which is our way of implementing the $\mathcal{M}^{FSP}$ machine, as illustrated in Fig. 7. In Section 6 we present the protocol in a formalized way and describe it in detail, and in Section 6.4 we bring accurate and efficient algorithms for the protocol's 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 FSP's protocol involves setting the credits each given by each FSP to its peers. As already explained, the credit that FSP A gives to its peer B is the amount that A risks trusting B with, i.e. it sets a limit to the loss A would incur if B behaved improperly. To set the credit for a peer $< f, k >$, the administrator instruction $(\texttt{give\_credit}, f, k, c, r, b)$ is used, where c is the credit given, 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, that set a limit to the rate of communications sent to this peer. The implementation is shown in Algorithms 8, 10.

5.2 PC and PO

The next stage of the protocol deals with issuing payment certificates (PC). Generally speaking, a PC is a commitment to pay money against specific payment orders (PO) under certain conditions.

Definition 5 *We define a PC as the tuple:*

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

The meanings 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 FSP identifiers, marking the path through which a payment passes.
- max is the maximum amount that should be paid under this PC.
- $expire$ is the expiration date of this PC.
- trt is the time it takes to handle a redemption under this PC.
- net is a commission function of redeeming against this PC; 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 limit the rate at which the peer by may serve redemption requests under this PC.
- v is a key for validation of a signed payment order.
- sig_{by} is the signature of the issuer on the PC.

Algorithm 4: FSP_Valid_ProtocolLog($ProtocolLog$)

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)
$Credit[n][] = 0$;
$Delay[n][] = 0$;
foreach $line(u, t, line\_type, v, paramters) \in ProtocolLog$ **do**
    // we go over ProtocolLog lines sequentially, u must be honest
    if $line = (u, t,$ give_credit$, v, k, credit, b, r)$ **then**
        // We allow only one give_credit instruction for each party
        if $Credit[u][v]! = 0$ **then**
            return True;
        // We allow only non-negative credit
        if $credit < 0$ **then**
            return True;
        $Credit[u][v] = credit$;
        // We allow LB rate no higher than the communication rate
        if $(b < 0) \vee (r < 0) \vee (r < R)$ **then**
            return True;
        // max delay from u to v
        $Delay[u][v] = L + b/R$;
        $Balance[u][v] = 0$;
    if $line = (u, t,$ valid_redemption$, v, id, amt)$ **then**
        $Balance[u][v] += amt$;
    if $line = (u, t,$ redeem$, v, id, amt)$ **then**
        $Balance[u][v] -= amt$;
        // Every valid redemption to an honest peer must be validated.
        if $(v \in H) \wedge (!\exists t_2 \leq t + Delay(u, v)) \mid$
        $(\exists (v, t_2,$ valid_redemption$, u, id, amt) \in ProtocolLog \vee \exists ($finish_game$, t_2) \in ProtocolLog)$ **then**
            return False;
    if $line = (u, t,$ payment_received$, v, amt)$ **then**
        if $Balance[u][v] + amt > 0$ **then**
            return True;
        $Balance[u][v] += amt$;
    if $line = (u, t,$ payment_certified$, v, amt)$ **then**
        $Balance[u][v] -= amt$;
    // Credit bounds the negative balance
    if $Credit[u][v] < -Balance[u][v]$ **then**
        return False;
return True;

PC flavors

Definition 6 *We distinguish between two PC flavors:*

1. Prime *PC: A PC issued independently at the instruction of the upper layer, together with its terms and conditions .*
2. Derived *PC: A PC issued at the instruction of the upper layer, based on a PC received from another peer. We refer to the PC on which the derived PC is based as a* Base *PC.*

Definition 7 *We define a 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 FSP identifiers, that form the path through which a payment should pass.
- *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 for accepting *PO*s. When a party accepts a *PO* redemption request it updates the balance for the sending peer, and returns a certificate notifying that the PO has been redeemed. Following are the conditions under which *PO*s are accepted with respect to a *PC*. Let $ACCEPTED(PC)^t$ denote the set of the *PO*s that were accepted with respect to a *PC*, by *PC.by*, prior to 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 state that a payment order PO is acceptable with respect to a 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 with an identical PO.id was accepted earlier with respect to this PC .*
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 to issue a PC has different parameters, depending on the PC flavor. We mention again the PC attributes:

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

1. In order to issue a prime PC, we use

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

This yields the following PC:

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

2. In order to issue a derived PC, from a 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 add another parameter, the base PC index in the PC issuer machine database (6.3). This would yield the following PC:

$$PC_1 < id_1, by_1, for_1, for_1||path_0, expr_1, trt_0 + Dalay(by_1, by_0), max_1, net_1, b_1, r_1, v_0, sig_{by_1} >$$

When an FSP issues PC_1, a credit of at least max_1, for by_0 is required. Otherwise, if a matching PO is redeemed, by_0 will owe by_1 a higher amount than the maximum allowed one. Moreover, if other PCs are derived from by_0 PCs, the sum of these PCs max attributes should not exceed the credit given to by_0. An FSP may issue several derived PCs based on the same base PC_0, but every PO that is redeemable against the derived PC should be redeemable against the base PC_0, to ensure that the derived PCs issuer does not lose money.

For the derived PC flavor, the leaky bucket allocations should be validated, so the issuer does not commit to a higher rate than it is able to handle. To this end, the protocol stores the maximum b and r that limit the communication for each peer. It also stores the available b and r of the base PC, and every time a derived PC is issued, it takes the given b and r parameters off the original b and r allocations.

The time it takes to handle a derived PC is calculated by adding the maximum delay to by_0 to the base PC handling time (trt_0). This makes it possible for the issuer to send a redemption request with a PO, and this redemption request should be received by by_0 before the PO expires. The implementation is described in Algorithms 17, 11.

5.4 Redeem PO

The next stage of the protocol is redemption. Every FSP keeps a balance for its peers, which changes whenever redemptions and payments are made. This balance is the amount this FSP owes to its peer. When an FSP receives a message $(\texttt{redeem}, PO, id, PO_{ind})$ from its peer, it verifies that the PO is acceptable with respect to the relevant PC ($PC.id = id$), according to the conditions listed earlier. If the PO is acceptable, the FSP responds with a signed message $(\texttt{redemption\_certified}, PO_{ind}, PO, time)$ and increases the balance for that peer by $PO.amt$. Once this peer receives this certificate it takes the said amount off its own balance. Of course, if the PO is accepted with respect to a derived PC, the PO must also be acceptable with respect to the base PC, and the FSP should send a redemption reqest with the same PO to the base PC issuer. In the present work we only consider one possible path for each PO redemption; this path is specified in the PO itself. The implementation is described in Algorithms 13, 14.

5.5 External Payment

The last stage of the protocol is the actual payment. The payment is external to our system, and is notified by the machine administrator. When an FSP receives notification $(\texttt{payment\_received}, peer, amt)$, where amt is the amount paid, it should increase the balance for $peer$ by amt, until it reaches zero (to simplify the protocol). The FSP must also send a signed certificate $(\texttt{payment\_certified}, peer, amt, time)$ to $peer$. Once $peer$ receives this certificate, it should take the sum off its own balance. The implementation is described in Algorithms 15, 16.

5.6 Communication Messages

Following is a list of messages that pass between the peers over the protocol:

- $(\texttt{given\_credit}, credit)$: This message informs the receiver that the sender gave it credit. When received, it is evaluated by Algorithm 10.
- $(\texttt{pc\_issued}, PC)$: This message includes a PC issued by the sender for the receiver. When received, it is evaluated by Algorithm 11.

- (`redeem`, PO, PC_{id}, PO_{index}): This message is used to request a redemption of a PO against a PC with id PC_{id}. It also includes the index of the PO in the sender database, to be referred to in the certification. When received, it is evaluated by Algorithm 13.
- (`redemption_certified`, $PO_{index}, peer, amount, time, signature$) - This message certifies that the sender accepts the PO sent by the receiver. Once received, it is evaluated by Algorithm 14.
- (`payment_certified`, $peer, amount, time, signature$): This message certifies that the sender has received an external payment in the amount of *amount* from the receiver. Once received, it is evaluated by Algorithm 15.

6 Protocol Descriptions

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

6.1 Notations

We outline briefly the definitions included in [3], and substitute a number of notations by more convenient forms. The symbol $\downarrow$ is the error element added to the domains and ranges of all the functions. We substitute it by the notation *null*. Generally speaking, 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 as $x.att$ 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. An entry x added to D is abbreviated as $D :\Leftarrow x$, we substitute it by $D : \mathbf{insert}(x)$. A database D of a machine owned by u is denoted by D_u. The set $\mathcal{INDS}$, isomorphic to $\mathbb{N}$, numbers all the entries in D consecutively. 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 implicitly. 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 of u machine is denoted by pks_u^{hnd}, and the public key that u saves for v is denoted by $pks_{u,v}^{hnd}$. We substitute these notations by $private\_key$, and $public\_key_v$, respectively, for u machine.

6.2 Cryptographic library abstraction

The algorithms are written in a pseudo code that supports the abstraction of the cryptographic library implementation, i.e. uses handles. A handle is a local name that the machine uses to reference cryptographic terms. We use the set $\mathcal{HNDS}$ to represent the handles. This mechanism allows a protocol description to be implemented both with an idealized cryptography and with real cryptography. In the idealized version the handles are the local names of Dolev-Yao-style terms; in the real version, they are the 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 is equivalent to the **list_proj** interface in the original notations.
- **store**, **retrieve** inserts and receives data from the cryptographic library using handles.

In the algorithms, the superscript *hnd* on a parameter denotes that it is a handle. We use the cryptographic library interfaces only with handle parameters. For example, we call the function *store* to get a handle of a normal parameter before signing, and *retrieve* is called before database insertions.

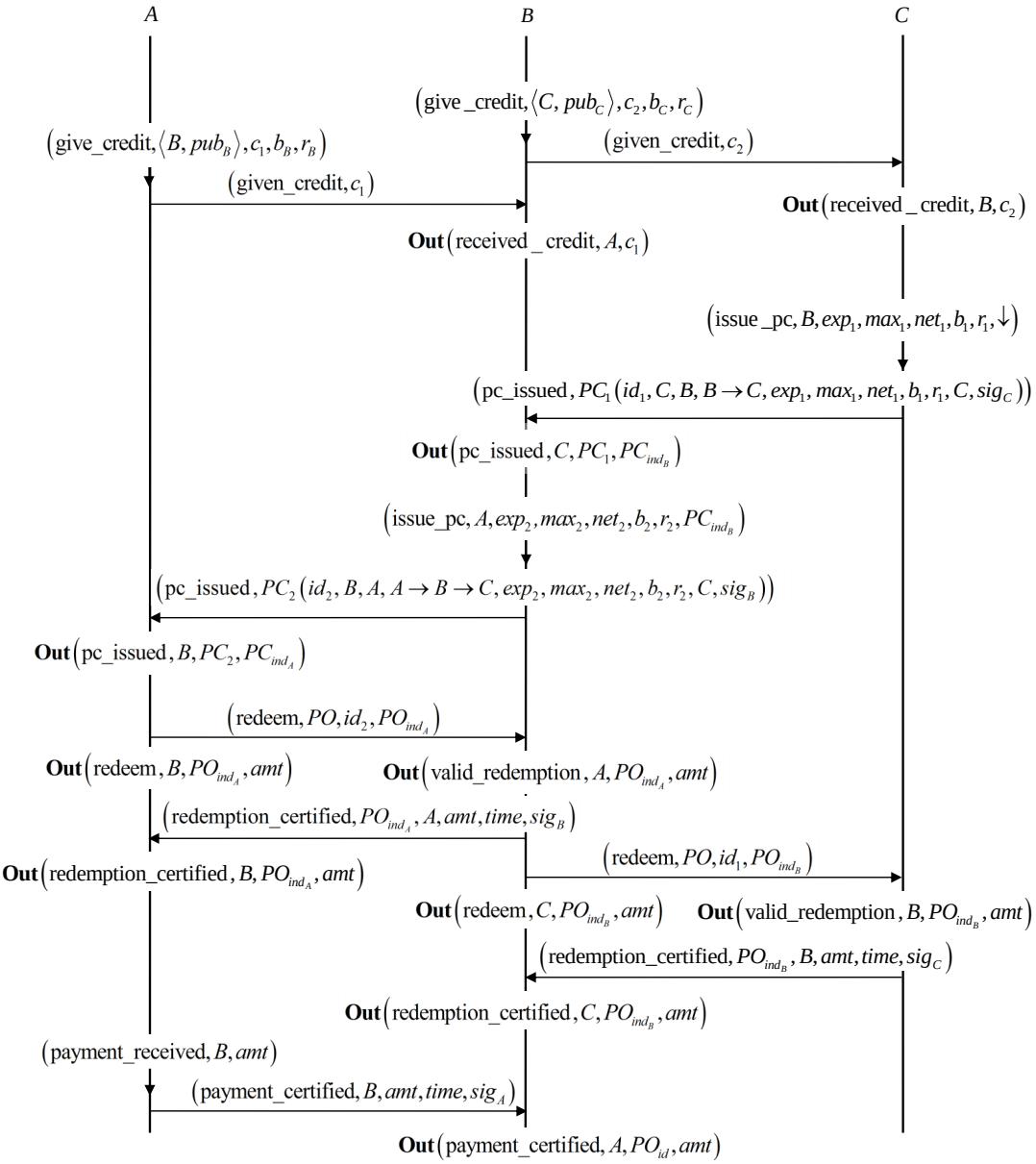

Fig. 7. Protocol Overview Illustration

6.3 Databases

The 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 a certain 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 maximum 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 to verify the leaky bucket transmission rate limitation according to the parameters allocated for sending messages to $x.peer$; t is the last update time, and x is the last bucket value.

Another database is D^{PC} , used to handle PCs. Each entry x in this database 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 meanings for a certain 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 defined above in Definition 5.
- $x.base$, in case of a 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 that u has not used yet.
- t, x are used to verify the leaky bucket transmission rate limitation according to the parameters allocated for sending redemption requests against this PC; t is the last update time, and x is the last bucket value.

Another database, D^{PO}, is used to handle 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 meanings for a certain peer u:

- $x.ind \in \mathcal{INDS}$, index in the database.
- $x.id, issuer, path, amt, expire, sig$, PO attributes as defined earlier.
- $x.pc \in \mathcal{INDS}$, 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 overview of the protocol is given in Fig. 7. In order to prevent message forgery, the sender must always sign before sending a message. The receiver must verify this signature every time an input arrives from the communication layer. To this end, we use the function '*sign_send*' (Algorithm 6) for all messages exchanged between peers. This algorithm is also an excellent example for working with handles. We first use the '*store*' and '*list*' function to prepare the parameters for signing (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 argument list of which the message is made, 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 itself by 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 ≠ null) ∧ ( verify_LB(D_u^FSP, D_u^FSP[(peer = f)].ind) ) = True) then
3      type^hnd ← store(title) ;
4      l^hnd ← list(title^hnd) ;
5      foreach param ∈ parameters do
6          param^hnd ← store(param) ;
7          add_to_list(l^hnd, param^hnd) ;
8      s^hnd ← sign(private_key, l^hnd) ;
9      m^hnd ← 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 ← 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 ≥ 0) ∧ (r ≥ 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 ← 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})

$l_j^{hnd} \leftarrow$ `parse_list(`l^{hnd}, j`)` for $j = 1, 2$;
if `verify(`$l_1^{hnd}, public\_key_f, l_2^{hnd}$`)` $=$ True **then**
  $type^{hnd} \leftarrow$ `parse_list(`$l_1^{hnd}, 1$`)` ;
  $type \leftarrow$ `retrieve(`$type^{hnd}$`)` ;
  for $i = 1$ *to* `list_length(`l_1^{hnd}`)` **do**
    $param^{hnd} \leftarrow$ `parse_list(`l_1^{hnd}, i`)` ;
    $param \leftarrow$ `retrieve(`$param^{hnd}$`)` ;
    `add_to_list(`$parameter\_list, param$`)` ;
  if $type =$ given_credit **then**
    `evaluate_given_credit(`$f, parameter\_list$`)` ;
  else if $type =$ pc_issued **then**
    `evaluate_pc_issued(`$f, parameter\_list, l_2^{hnd}$`)` ;
  else if $type =$ redeem **then**
    `evaluate_redeem(`$f, parameter\_list$`)` ;
  else if $type =$ redemption_certified **then**
    `evaluate_redemption_certified(`$f, parameter\_list, l_2^{hnd}$`)` ;
  else if $type =$ payment_certified **then**
    `evaluate_payment_certified(`$f, parameter\_list, l_2^{hnd}$`)` ;

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

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

Algorithm 11: evaluate_pc_issued(f, pc)

$(id, by, for, path, expire, trt, max, net, b, r, v, sig) \leftarrow$ `parse_list` (pc);
$j := D_u^{PC}[(pc\_id = id) \wedge (pc\_by = by)].ind$;
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`
$(net(max) \leq D_u^{FSP}[peer = f].credit) \wedge$ `// net amount does not exceed the credit for the base pc issuer`
$(D_u^{FSP}[peer = f].b \geq b) \wedge (D_u^{FSP}[peer = f].r \geq r)$ **then**
  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)`
  $D_u^{FSP}[peer = f].credit- = net(max)$;
  $D_u^{FSP}[peer = f].b- = b$;
  $D_u^{FSP}[peer = f].r- = r$;
  `Output((`pc_issued$, pc$`))` ;

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

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

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

if `is_acceptable`$(po, pc\_id, now())$ **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)`
    $D_u^{PC}[pc\_id].avl- = D_u^{PC}[pc\_id].net(po\_amt)$;
    $D_u^{PC}[pc\_id].x+ = 1$;
    $D_u^{FSP}[peer = f].balance+ = D_u^{PC}[pc\_id].net(po\_amt)$;
    `sign_send`$(f,$ redemption_certified$, po_index, f, D_u^{PC}[pc_id].net(po_amt), now())$;
    `Output`(valid_redemption$, f, po_index, D_u^{PC}[pc_id].net(po_amt))$;
    $base\_id := D_u^{PC}[pc\_id].base$;
    if $(base\_id \neq null) \wedge (D_u^{PC}[base\_id].ind \neq null)$ **then**
        $p := D_u^{PC}[base\_id].by$;
        $D_u^{FSP}[peer = p].balance- = D_u^{PC}[base\_id].net(po\_amt)$;
        $D_u^{PC}[base\_id].x+ = 1$;
        `sign_send`$(p,$ redeem$, po, base_id, cur_ind^{PO})$;
        `Output`(redeem$, p, cur_ind^{PO}, D_u^{PC}[base_id].net(po_amt))$;

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

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**
    $D_u^{PO}[po\_ind].cert :=$ `list`(redemption_certified,*f,po_ind,for,amt,time,sig*) ;
    `Output`(redemption_certified$, f, po_ind, amt)$;

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

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

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

Input: (`payment_received`, f, amt)

if $D_u^{FSP}[peer = f].balance + amt \leq 0$ **then**
  $D_u^{FSP}[peer = f].balance += amt$;
  `sign_send`(f, payment_certified, $f, amt, now()$) ;
  return success;
else
  return failure;

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

Input: (`issue_pc`, $f, expr, max, net, x, b, r, base\_ind$)

if ($base\_ind = null$) **then**
  `// prime PC`
  $path \leftarrow$ `list(`f,u`)` ;
  $pc \leftarrow$ `list(`$cur\_ind^{PC}, u, f, path, expr, 0, max, net, b - r * Delay(u, f), r, public\_key_u$`)` ;
  $sig \leftarrow$`sign(`$private\_key, pc$`)` ;
  $pc \leftarrow$ `list(`pc, sig`)` ;
  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)`
  `sign_send(`f, pc_issued, pc`)` ;
  return success;
else
  `// derived PC`
  $p \leftarrow D_u^{PC}[base\_ind].by$;
  if ($base\_ind \leq cur\_ind^{PC} \wedge p \neq u$) `// base PC exists`
  $\wedge(expr \leq D_u^{PC}[base\_ind].expr - Delay(u, p))$ `// pc expires before its base PC`
  $\wedge(D_u^{PC}[base\_ind].net(max) \leq D_u^{PC}[base\_ind].avl)$ `// max amount does not exceed the base PC`
  $\wedge(b \leq D_u^{PC}[base\_ind].b) \wedge (r \leq D_u^{PC}[base\_ind].r)$`// committed rate not higher than the base PC`
  then
    $D_u^{PC}[base\_ind].avl- = D_u^{PC}[base\_ind].net(max)$;
    $D_u^{PC}[base\_ind].b- = b$;
    $D_u^{PC}[base\_ind].r- = r$;
    $path \leftarrow$ `list(`$f, D_u^{PC}[base\_ind].path$`)` ;
    $trt \leftarrow D_u^{PC}[base\_ind].trt + Delay(u, p)$;
    $pc \leftarrow$ `list(`$cur\_ind^{PC}, u, f, path, expr, trt, max, net, b, r, D_u^{PC}[base\_ind].v$`)` ;
    $sig \leftarrow$`sign(`$private\_key, pc$`)` ;
    $pc \leftarrow$ `list(`pc, sig`)` ;
    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)`
    `sign_send(`f, pc_issued, pc`)` ;
    return success;
  else
    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;

7 Security Analysis

7.1 The cryptographic library

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

$$Sys^{cry,id}$$

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

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

$$Sys^{cry,real}$$

7.2 The Theorems

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

Definition 1 *We state 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 of*

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

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 the following claim in ([3], Theorem 1 (Security of Cryptographic Library): *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.* In their notation, they proved that:

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

That is, any polynomial attacker able to break the system using the real cryptographic library, with non-negligible probability, is able break the system when it uses the ideal one. According to Theorem 1, the Inter-FSP Funds Transfer Protocol is secure with $Sys^{cry,id}$. From the Backes et al. theorem, it follows that the 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 $Delay(u, v)$ limits the communication delay of a redemption request that passes 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, v should receive it at time t_2 s.t. $t \leq t_2 \leq t + Delay(u, v)$.*

We claim that the credit limits a 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 the upper layer of u, where $c > 0$.

Claim 3 *For every honest peer u,*

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

at every step of running the game.

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

Claim 4 *If an honest peer u receives a payment order PO from its peer v at time t, and the PO is acceptable at time t, with respect to a 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 attributes of PCs they have 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 to another 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. Let v denote $x.by$, and let po be a certain 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 a derived PC (Definition 6), is also 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 a payment certificate pc_base received from p. Every payment order that is acceptable with respect to $pc\_derived$ at time t, is also 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 must 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 in lines 19 and 27 are never satisfied. We start with the condition in line 27:

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

It is clear that $Credit[u][v]$ is equivalent to $Credit(u, v)$. Claim 1 also maintains 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 $FSP\_Valid\_ProtocolLog$. Therefore, it follows from Claim 3 which maintains $Credit(u, v) \geq -D_u^{FSP}[peer = v].balance$, that this condition is never satisfied. We now show that the condition in line 19, is never satisfied. To this end, we must 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 $t_2 - t_1 \leq Delay(u, v)$. Note 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 an output can only occur in Algorithm 20. in line 9 and in Algorithm 13. in line 14:

1. Algorithm 20. line 9. Algorithm 20 is the evaluation of a $(\texttt{po\_received}, po, pc\_id)$ instruction, of a payment order received from the upper layer. To reach line 9, the conditions in line 3 must be satisfied; in particular,

   $$is\_acceptable(po, pc\_id, t_1 + Delay(u, v)) = true$$

   must hold. 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 ends before a redemption message was received, 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 an 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 mentioned earlier, the condition in Algorithm 20 in line 3 must be satisfied in u. This means that there is a record x in D_u^{PC} where $x.id = pc\_id$, and $x.by = v$. In this case, according to Claim 6, if

   $$is\_acceptable(po, pc\_id, t_1 + Delay(u, v))$$

   returns true in u, then

   $$is\_acceptable(po, pc\_id, t_2)$$

   must also return true in v, and v should output $(\texttt{valid\_redemption}, u, id, amt)$. This output leads 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 in line 1 must be satisfied, i.e the input of payment order *po* was acceptable at time t_1 with respect to the payment certificate PC that is stored in the record $D_u^{PC}[id = pc\_id]$. To reach line 14, the condition in 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 redemption request message to v, which is $PC\_base.by$. According to Claim 2, this message will reach v no later than $t_1 + Delay(u, v)$. Of course, if the game ends before the redemption request message arrives, 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 also acceptable with respect to the base PC at time $t_1 + Delay(u, v)$.
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 leads to the required $(v, t_2, \texttt{valid\_redemption}, u, id, amt)$ record in *ProtocolLog*.

■

7.5 Proof of the Claims

In order to prove our 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 time a record is inserted into D_u^{FSP} is in Algorithm 8. line 2, and the condition in 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 a redemption message to its peer v, then u must have exactly one record $D_u^{FSP}[peer = v]$.*

Proof. The only places where u is able to send a `redeem` message are in Algorithms 20 and 13. In both places we check first that a payment certificate from v exists in D_u^{PC} (in line 3 and in line 9, respectively). Such a payment certificate may be inserted into D_u^{PC} only in Algorithm 11 at line 6. To reach this insertion, the condition in line 3 must be satisfied; particularly, $D_u^{FSP}[peer = v] \neq null$ must hold. 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 a redemption against a payment certificate PC unless the rate meets the LB limitation with $PC.b, PC.r$ as parameters.*

Proof. The only places where u is able to 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 returns true only if the condition in 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 this function is called, 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 represents the size of the bucket ofpending 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 passes 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 is validated against $PC.b, PC.r$. The x attribute of this record is incremented with each request (Algorithm 13 line 4, and Algorithm 20 line 7). In Algorithm 13, u sends the redemption of a payment order that was accepted with respect to a 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 payment certificates derived from *base_pc*, does not exceed $pc\_base.b, pc\_base.r$, respectively. To issue a derived payment certificate Algorithm 17 is used. In this algorithm, the derived payment certificate b, r attributes are checked in line 14, against the attributes b, r in the *pc_base* record

of D_u^{PC}. These attributes are initialized to $pc\_base.b, pc\_base.r$ once pc_base is received, in in line 6 of Algorithm 11. For every PC derived from pc_base these attributes are reduced by the b, r of the derived PC in lines 17 and 18 (When the derived PC is timed out these attributes are increased by the same values of 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 a certain PC, is limited by the total maximum amount of the PC.*

Proof. An honest peer u, saves in avl the attributes of record x in D_u^{PC} the amount remaining in the payment certificate that is stored in this record. This attribute is initialized to the $x.net(x.max)$ when the payment certificate received is stored (Algorithm 11. line 6). A redemption can only be sent 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 in line 6. In Algorithm 13, u would redeem a 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 the `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 in 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 in line 16 (and with the update of timeout PC in Algorithm 18, line 4), it limits the total maximum 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 maximum 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 the sake of intelligibility, 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 in line 6, that is, $(\texttt{PC\_issued}, pc\_base)$ 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 mentioned above, pc_base was received from the communication layer as a $(\texttt{PC\_issued}, pc\_base)$ message from p. The verification of the signature is done in Algorithm 9, line 2 before this input is evaluated 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 $pc\_derived.path$. Therefore, $pc\_base.path$ 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 condition, 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, then $t \leq po.expr - pc\_derived.trt$ and $t \leq pc\_derived.expr$.
   In line 12 it is checked that $pc\_derived.expr \leq pc\_base.expr - Delay(u, p)$. Therefore, $t + Delay(u, p) \leq pc\_base.expr$. Also, according to line 20, $pc\_derived.trt = pc\_base.trt + Delay(u, p)$. Therefore, if $t \leq po.expire - pc\_derived.trt$, then $t + Delay(u, p) \leq po.expire - pc\_base.trt$.

6. *No other PO was accepted earlier with respect to this PC, with an identical PO.id.* Of course, *po* is acceptable only once. Every PO has a specific redemption path, so if a *po* is found to be unique in u, it will be unique in p as well, since no other peer is able to redeem this 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 the *pc_base* record in D_u^{PC}. According to Sub-Claim 4, this condition limits the net amount of all the redemptions against *pc_base*, including redemptions of payment orders that were accepted with respect to derived PCs 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 the b, r parameter of all the payment certificates derived from *base_pc* does not exceed $pc\_base.b, pc\_base.r$, respectively. This is verified in line 14, with the attributes r, b in the *pc_base* record in D_u^{PC}. These attributes are initialized to $pc\_base.b, pc\_base.r$ when *pc_base* is received, in Algorithm 11 at line 6. For every PC derived from *pc_base*, these attributes are reduced by the b, r of the derived PC in lines 17 and 18 (When the derived PC is timed out Algorithm 18 lines 5,6) increases these attributes by the same values. This way, if *po* is acceptable with respect to *pc_derived*, it is 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 $pc\_base.r, pc\_base.b$.

Therefore according to Definition 8, the claim holds. ■

Sub-Claim 6 *The return value of the function* $is\_acceptable(PO, pc\_id, t)$*, when 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 in Algorithm 12. and returns true only if the condition that ends in line 9 is satisfied. We repeat Definition 8 that specifies the constrains that should hold for a payment order *PO* to be acceptable with respect to payment certificate *PC*, and for each constrain show that it holds, by referring to the relevant lines in Algorithm 12. For the sake of intelligibility, we rewrite the definition and add our proof of each part in line:

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 a record can only be inserted in Algorithm 17 at lines 6 or 24 respectively, and only if a prime PC or a derived PC is issued. Immediately 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 a record can only be inserted in Algorithm 11. that evaluates receiving a payment certificate from the communication layer, at line 6. According to the condition in 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 a *PC* is u itself, it should sign the *PC* and store the signature in *PC.sig* in Algorithm 17 lines 7, 25. Otherwise, if a *PC* is 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 has been accepted earlier with respect to this PC, with an identical PO.id.* This is checked in line 4. This condition checks that there is no record in D_v^{PO} from the same issuer, with an *id* attribute that is identical to *PO.id*, given that the payment order *id* attribute is unique for each issuer (Algorithm 19). A payment order record can be inserted into D^{PO} only after successful check of *is_acceptable*() with this payment order (Algorithm 13. line 2, Algorithm 20. line 4). It follows that every payment order can pass *is_acceptable*() only one time per machine, when a redemption is sent or received.

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 limits the net amount of all redemptions made against a 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 that a single message sent at the time this function was called, 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 represents 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. The $x.x$ is incremented by each request (Algorithm 13 line 4, and Algorithm 20 line 7).

■

Proof of Claim 1

Claim 1 *When $FSP\_Valid\_ProtocolLog$ (Algorithm 4.) is run 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 in line 2, following instruction

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

to machine u; 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 an 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 in 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 where $D_u^{FSP}[peer = v].balance$ 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 all the possible changes 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 an 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 an 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 an 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 an 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 be received by v at time t_2 s.t. $t \leq t_2 \leq t + Delay(u, v)$.*

Proof. Note that $Delay(u, v)$ (Definition 3) limits the delay of communication between u to v as long as the underlying communication layer guarantees sending requests at a certain service rate defined in the LR server model by the parameters L, R. This is given by the assumption of
$$LR\_FIFO\_Valid\_CommLog(CommLog) = True$$
In addition, the rate by which messages are sent 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 the transmission rate of an honest party meets this limitation. To this end 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 of Algorithm 8, which evaluates the

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

instruction. A record may be inserted according to the condition in line 1 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 determine that the total sums of all b, r attributes of the stored payment certificates from v never exceed the given initial b, r parameters in the `give_credit` instruction. Whenever u receives a payment certificate PC from v, the values of $PC.b, PC.r$ are verified to be lower than $D_u^{FSP}[peer = v].b, D_u^{FSP}[peer = v].r$ respectively (Algorithm 11, line 5), and if a 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. $D_u^{FSP}[peer = v].b, D_u^{FSP}[peer = v].r$ 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 never sends a redemption against a payment certificate PC unless the transmission rate meets the LB limitation with $PC.b, PC.r$ as parameters. The only way for an honest peer to send a message, is by using the *sign_send*() function (Algorithm 6). The LB is limited in Algorithm 6, at line 2, using the function $verify\_LB(DB, ind)$ (Algorithm 5). Only messages that are not redemptions are verified, as the redemption messages rates are pre-allocated, and the verification is done against the payment certificate, as already mentioned.

The function $verify\_LB(DB, ind)$ verifies that sending a single message at the time this function is called 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 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 allocation of the rates of the redemptions. 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 each time a message of any type is sent (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 for its peer in $D_u^{FSP}[peer = v].balance$ is the amount that u owes to v, that is, when this is a negative value, v owes money to u. Obviously, the $Credit(u, v)$ is always non-negative, by definition. We need to show that the balance is never lower than $-Credit(u, v)$, even if it is negative.

According to Claim 1, $D_u^{FSP}[peer = v].balance$ is initialized to zero, and decreases only when an external payment is certified and a redemption request is sent. On certification of an external payment (Algorithm 15), the balance never goes below zero, according to the condition in line 1, so it is still higher than any negative value. Therefore, the only possible way for the balance to have a negative value is by sending a redemption to v, in which case the balance decreases by the net amount of the redemption. An honest peer u saves the remaining credit in $D_u^{FSP}[peer = v].credit$ for its peer v. This attribute is initialized to $Credit(u, v)$ in Algorithm 8 at line 2. Every time a payment certificate is received from v, it is checked and updated that the net maximum amount that can be redeemed using the received payment certificate is lower or equal to $D_u^{FSP}[peer = v].credit$, and is reduced by the same value (Algorithm 11 lines 4 and 7). When a PC is timed out, $D_u^{FSP}[peer = v].credit$ is updated and is increased by the same value (Algorithm 18. line 9). According to Sub-Claim 4, all the redemptions made against a certain payment certificate are limited by the maximum net amount of this payment certificate, and the total sum of all the payment certificates of a certain peer, is limited by this peer's initial credit. ■

Proof of Claim 4

Claim 4 *If an honest peer u receives a payment order PO from peer v at time t, and the PO is acceptable at time t with respect to a certain 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 reception of a payment order in Algorithm 13, and outputs the required message $(\texttt{valid\_redemption}, v, PO.id, PC.net(PO.amt))$ if the condition in 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. Therefore, given that PO is acceptable at time t, with respect to 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. in 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 into 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. If a message is sent from an honest v, the only possible place to send this message is in Algorithm 17 at lines 7,25. To reach one of these lines, v must insert a record whose 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. Let v denote $x.by$, and let po be a certain 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.

Proof. The implementation of the function $is\_acceptable()$ is in Algorithm 12. and it returns true only if all the conditions in lines 2 - 9 are satisfied. We list below 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 an attribute that equals 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 depends only on the parameter *po*, which is the same when it is run in u. Therefore, if it is satisfied in u it is also satisfied in v.
3. Line 4: This condition checks that there is no record in D_v^{PO} with an *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 (Algorithm 13. line 1, Algorithm 20. line 3). A payment order record can only be inserted into D^{PO} 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. Therefore, every payment order can pass *is_acceptable*() only one time per machine. Once the condition in line 9 (specific redemption path for a PO) is satisfied in u, a payment order can pass $is\_acceptable(PO, pc\_id, time)$ in v, only if it comes from u. Therefore, when u runs *is_acceptable*, if there is no record in D_u^{PO} with *id* and *issuer* attributes that equals *po.id* and *po.issuer* respectively, then when v runs it, there will be no record in D_v^{PO} with *id* and *issuer* attributes that is equal 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 showed above. We will show that $D_v^{PC}[pc\_id].avl \leq D_u^{PC}[pc\_id].avl$ by induction. In both records, the attribute *avl* is initialized to the attribute *max* when it is inserted into 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 issues a 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 the redemption of a *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 above, such a *PO* can be received only from u. Therefore, if v reduced $D_v^{PC}[pc\_id].avl$ by the net amount of a certain *PO* from u, u should reduce from $D_u^{PC}[pc\_id].avl$ an equal or greater amount, because the net amount of the payment orders that come from a derived payment certificate are limited by the *max* attribute of the derived payment certificate, and the net amount of a payment certificate that comes from the upper layer is equal on both sides.
5. Line 6, validation of the redemption rate. LB is validated using the function $verify_LB()$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 a payment order received from u (Algorithm 13. line 4), and is 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-hand sides of the inequalities are the same when run by u and by v, according to Claim 5. The only difference is in the *time* parameter. Note that u runs *is_acceptable* with *time* parameter is equal 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-hand sides of the inequalities, when run by v, will be equal or lower from those run by u, meaning that the conditions are satisfied.
7. Line 9. This condition depends on a *po*, which is the same when run by v as by u, and on $D_v^{PC}[pc\_id].path$ that according to Claim 5, $D_v^{PC}[pc\_id].path = D_u^{PC}[pc\_id].path$. Therefore, if it is satisfied for u, it is satisfied for v as well.

We showed that for each condition in *is_acceptable*, if it is satisfied in u, it is 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 a 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* $t_2 | t \leq t_2 \leq t + Delay(u, p)$.

Proof. An honest party can only issue *pc_derived* 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 payment order that is acceptable 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 the safety property of the Inter-FSP Funds Transfer Protocol. In the present section we prove the liveness property, i.e. that the protocol reacts to instructions without throwing exceptions. Actually, in most of the protocol's instructions, the protocol throws exceptions only for trivially invalid input. However, in the instructions `issue_pc` and `po_received`, it is not trivial to show that exceptions do not occur. We need to show that for these instructions, the Inter-FSP Funds Transfer Protocol only throws exceptions for invalid instructions.

Definition 10 *We define the expected behavior of $\mathcal{M}^{FSP}$ upon an* (`issue_pc`, $PC\_attributes$) *instruction, as follows: The machine should sign a payment certificate PC that comprises the given attributes, as defined in Definition 5, and send a* (`pc_issued`, PC) *message to $PC.for$. This should be done under the restriction that redemption of any payment order that is 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 a* (`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 another peer, and that the PO is acceptable with respect to it at the time the redemption is received.*

Definition 12 *We state that $\mathcal{M}^{FSP}$ has a 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. For each instruction, we show that the Inter-FSP Funds Transfer Protocol behaves as defined in Definitions 10 and 11:

1. The instruction (`issue_pc`, $PC\_attributes$) is evaluated in Algorithm 17. In case of a prime PC, the required payment certificate is sent at line 7. In case of a 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 PO that is acceptable 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 a PO that is acceptable with respect to its derived PC, and is not acceptable with respect to the base PC at the time the redemption is received, it will lose the money according to Definition 4. Therefore, the condition that ends at line 15 validates precisely the restriction of Definition 10.
2. The instruction (`po_received`, PO, PC_{id}) is evaluated in Algorithm 20. The `redeem` message is sent as required in line 8, only if the condition at line 3 is satisfied. This condition checks that the given PC was received by machine u from another 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. Therefore, the condition at line 3 validates precisely the restriction of Definition 11.

■

9 Conclusions

- Initially, we present the Inter-FSP Funds Transfer Protocol. The protocol ensures reliability by limiting the rate of service for each peer.

- The protocol was designed with a conservative risk limiting mechanism, ensuring that each of the parties meet its commitments even in worst case situations. Note that real systems often take higher risks to maximize revenues. In such cases a system with additional freedom degrees should be implemented based on our work. For instance, it is possible to limit the risk by letting each party estimate the probability of loss in addition to the credit it gives to its peers, and add a parameter that sets a limit to the probability of bankruptcy.
- The proposed protocol handles the relationships between the peers and the transactions they make. The problem of finding a path made of peers that trust and work with each other is out of this protocol's scope. The difference between these two tasks resembles the difference that exists in networking between an IP protocol and routing. In the future, work could be done on routing, i.e. determining an optimal path on a given FSP network.
- In cases that are considered as losses, adding an evidence layer to our protocol could provide sufficient evidence for disputing outside the protocol. This evidence should include non-repudiated delivery of messages and signed certificates that the peers give to each other (see [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.