Cryptographic Funds Exchange Protocol

GXAC2C4BC37CD6OON7CIDWAWNLIN6HHJ

Thomas Cerny
thomas.cerny@~~abrantix.com~~

September 21, 2014

1 Introduction

This paper introduces a construction designed to support the exchange of funds using a network of electronic devices. The construction is based on the fundamental set of accounting rules that reflect common sense and are accustomed for many centuries. It employes cryptographic methods, namely digital signatures and hash functions, as foundation for a family of protocols meant to model traditional and future habits related to transfer of funds. We shall present the foundation and some examples of interaction protocols here.

Distinctive features of the construction presented are:

- Decentralization, no authority required.
- Allows transfers between two parties without support from a (trusted) third party.
- Does not unwantedly leak or reveal information as part of funds exchange.
- Does not perform unnecessary calculations.
- Allows arbitrary security measures to be employed to protect assets.
- Does not require rounding.

Inspiration for this work comes from different sources such as existing (bank) exchange solutions and Bitcoin. While being considered inadequate, the latter has contributed valuable grounds for the ideas described here, and obviously, our construction must be contrasted with the former.

2 Principles

As first it is necessary to define the principles the construction described is based on. The principles shall be described in terms of *protocols* consisting of *messages* being exchanged between parties and in terms of *books* and associated *keys* used to store and generate such messages.

1. A protocol exists so that two parties can exchange funds without involving another party.
2. Protocols do not unwantedly disclose information about balances and transactions.
3. Protocols unambigously define how related messages are to be stored in the book.
4. Messages affecting the state of a book are signed by the key associated with the book.
5. Messages affecting the state of a book refer to the recent state of that book.
6. The balance of a book does never cross zero.

We expect the total amount of funds to sum up to zero over all books at any time, if all parties act correctly. Should this not be the case, at least one party is playing a false game. To deduce who, the books must be examined.

3 Account Lifecycle

We will use the term *account* to refer to a book and its associated secret key in the following. Having access to the secret of a book means the ability to write the book and thus to perform interactions with other parties. The related public key is used to verify the signatures of the messages in the book and their ordering. It seems worth to emphasise that the book and the key are two different physical objects and do not need to be stored at the same place.

The lifecycle of an account begins with creation of a new key pair consisting of a private *account secret* and a corresponding public key, the *account identifier*. It is assumed that cryptographic parameters and random number generators are choosen so that the creation of the same key pair at two different occasions is not feasible.

Manifest

The *first message* to be written to the book is the *manifest*. It contains information the creator is willing to disclose to the book and consequently its readers. Because the manifest is always the first message in the book, it does not explicitly refer to a previous state, as required by principle 5, but is implicitly tied to the initial state of the book. Any message subsequently written to the book must not be a manifest and must explicitly refer to a state of the book.

Balance

The *balance* of an account is a vector with the dimension of currencies having elements from the rational numbers. The initial balance of an account is zero. Principle 6 implies that at any time all elements in the vector must be either all positive or all negative.

Every message written to the book is unambigously associated with a protocol, which states precisely how the message contents affects the account balance in terms of algorithms that work in polynominal time on the balance vector and the message content. Simple algebraic expressions are preferred.

Linking

Protocol messages can refer to other messages and documents by including a hash value as a reference. It is assumed that the bookkeeping system is capable of storing such a map of compressed values to source documents so that references can be resolved whenever required. Some of the messages, namely those received from others are not part of the books ledger chain, but appear in the map and thus are integral part of the book.

Termination

Any account with balance zero can be closed, a final state from which it can not revert. The fact of closing an account is accompanied by the writing of the *termination* message into the book, which is the *last message* that may possibly appear in a book.

Ordering Requirements

Left to say that, in general, a message can also refer to more than one previous state of the book, for the case that some physically distributed instances of a single book are unable communicate to exchange their most recent states for some time and the bookkeeper wants to consolidate the split later. In other words, this construction does not require *total ordering* of the messages in the book. It only requires that at any time all histories desribed by the book obey principle 6.

4 Faithful Histories

So far we have not thought about all the things that could go wrong or could be manipulated. Of course these questions must be addressed. But before doing so, let some more general considerations preceede:

- The majority of people is willing to obey rules.
- Its stupid to break rules and leave evidence.
- Exposing information can relief from accusation.
- Good rules give good systems (many examples).

Now lets think about an attack: consider a case somebody breaks principle 6, letting his account cross the zero line from the top, virtually giving credit to himself. This fact would clearly be visible in the book of that account, so if ever asked to disclose his book, the fraudster will face a serious problem in that he can only prove his guilt, but not the opposite. So he could as well forget the private key and pretend loss. It both comes to the same end, either innocence is proven or guilt is assumed[1].

Moreover the recipients of fraud funds will face the problem that there is nobody to reimbourse them. It's their own fault they accepted these funds by adding them to their book without checking the sender and the best to get away with it is to loose these funds.

Of course the recipients might have transferred funds to other recipients while the fraud was going on. These parties will be affected as well by the fraud, the damage will spread and everyone affected will guess where it came from.

It should be clear that this construction puts responsiblity into the hands of the recipient. This does not apply to fraudulent accounting only but to inacceptable sources in general. In practice, nobody is receiving funds spontaneously without a related business case. Most countries require their citizens to reveal the sources of their income, equivalently asking for some content from the book.

From this and the principle 1 we conclude that while two party transfer is possible, it may not be the prominent use case. Instead the recipient will want to have some assurance that a particular account has a *faithful history*, which could be given by a third party without disclosing any of the book to the recipient.

The motivation for principle 1 is not so much its application but its availability. Two parties willing to exchange can always do so, if they are ready to take the risk of being cheated by the opposite party. It just keeps freedom where it is today.

Please note that this construction is based on some assumptions that are not technical by nature, in the sense of other systems employed to exchange funds electronically. This reliefs the construction from requiring a hierarchical, centralized structure but carries the risk of instability, should assumptions implied by the principles be circumvented by, for example, large organizations.

It however leaves, at every node of the exchange graph resulting from the combined histories the option to ask for the history of the parties involved. Every party may deny such a request, depending on how loud the question is being asked. This will eventually lead to the recovery of some of the graph, depending on the level of cooperation provided by the account holders. It is not difficult to imagine that a govern can motivate citizens to open their books by a wide spectrum of measures.

[1] the recipient is obliged to keep the book available to defend himself, as usual in these matters

5 Properties

This section shall explain some of the properties of the construction described and systems built out of it. So far we touched the question of how responsibility can be distributed among the parties sharing and using a common means of value representation, a currency. The term *currency* will be discussed in more detail later. Let's look at it as something arbitrary abstract but of comparable value given a unique name.

The protocols described herein aim to establish a means by which funds, given as values denoted in some currencies, can be exchanged between arbitrary parties. The recipient may request the sender to present a prove that his account is faithful or impose other requirements to protect himself from fraudulent, illegal or unwanted funds. In general protocols will include a token derived from the current state of the book to tie account histories together.

At any time each party shall be ready to provide information on her book if requested so by a legitimate other party in order to resolve an issue related to one of the parties accounts. We do not discuss how such legitimation is achieved in practice, but it is certainly an important topic[2]. Any party can deny the request from a non-legitimate party, but may also choose to provide it anyway.

It is important to understand that the construction described here, due to its distributed, decentalized nature, may show unpredicable behaviour and instability if exposed to certain conditions, such as destruction of portions of the books. It is also, per se, manipulation agnostic in the sense that it can not distinguish, by intrisic means, a faithful from an unfaithful account and thus must leave that question to another layer.

But most importantly, this construction is not designed to prevent fraud altogether, as this requires centralization and authority what in turn violates principle 1. Instead it leaves it to interacting parties to figure out what to do to trust each other. This ultimately requires a decision from a human beings as the account owners, unless routed by convenience rules. Consequently this construction is not based on the functioning of a central entity, particular calculation or other technological procedure only, but also on the reasoning of human beings[3].

To guide that reasoning, the construction provides a powerful instrument to those who act correctly[4] when it comes to defend themselves. At the same time it provides a means to investigate fraud should irregularities be detected, to any extent the investigator is able to read the books, but without exposing the whole system, unless the whole system is affected.

The construction does not preclude extreme configurations. For example, by requiring all citizens of a country to disclose their expenses[5] (this means to disclose all transfers they confirmed), *freedom* could be affected in some sense. A completely uncontrolled configuration could, on the other hand, lead to built-up of massive ponzi pyramides and subsequent losses.

The construction *does not ensure the correctness* of any account, transaction or balance. It merely provides the faithful party a means to unambigously describe and commit to a *notion of truth* that she can later use to prove the correctness of her actions.

[2]already highly regulated by existing laws
[3]as a primary ingredient in contrast to other systems
[4]another legal question
[5]they already must disclose their revenues

6 Protocols

Every information exchange that involves data signed by one or more account secrets must be considered a part of some protocol of interest if the exchange has the potential to change the balance of one or more accounts. This means that, theoretically, protocol elements can be scattered around across different messages and possibly may be embedded in other applications communication.

While the choice of protocols is ultimately done by the parties applying them, some basic interactions shall be presented in this section.

There is no limit in the number or kind of protocols or applications employing the account secret in one way or another implied by this construction. It shall however be stated that correct protocol design and definition are crucial to the operation of the construction. According to principle 4, every protocol must unambigously declare how the messages contents is to be accounted for, in respect to the account balance, for every message to be written to the book.

6.1 Account Identifiers

The general definition of protocols does not depend on the exact choice of cryptographic algorithms if we restrict ourselfes to Diffie-Hellman type based on whatever suitable group, elliptic curves for example, given some generator element G for that group. The relation between the public account identifier A and the private account secret a is given by $A = aG$ where the group operation is written additively.

While the account identifier ultimately is defined to be the public key A, depending on the use case, a losfully compressed representation may be used for display purposes. Within protocols, account identifiers shall always be represented in full length of their encoded form.

6.2 Generalized Amounts

For the sake of generality we also assume that funds are represented as vector quantities, dimensions given by currencies and values given by rationals as

$$\mathbf{v} = (v_1, v_2, ..., v_n) = \sum v_k \mathbf{u}_k, \quad v_k \in \mathbb{Q}, \quad |\mathbf{u}_k| = 1$$

where $\mathbf{u}_k$ is a base vector of the currency space. In practice the vector is transferred as a map from a *currency identifier* to a rational number, zeroes left out.

6.3 References

As mentioned before, at some times a message contains one or more references to preceeding messages. Such a reference is given by the result of a suitable hash function applied to the encoded message.

6.4 Ledger States

Any message that refer to a state of the book shall include a set of preceeding states representing the most recent state of the book, as known to the actual party. In a sequential history, each message except the manifest refers to exactly one preceeding state of the book.

6.5 The Transfer Protocol

Given account A and B the transfer protocol performs the task of moving funds from A to B by means of essentially three messages, at the initiative of A. Two other messages are written to the book but not exchanged.

The messages used in this protocol share some common structure. Each message consists of a signed body part and a list of attachments, which may be signatures, certificates or other data not included in primary signature generation. The body must contain at least a message type identification and the account identifier of the primary signer. Besides that the signed body may contain data specific to individual message types as well as opaque, application specific extension data.

Individual protocol message may refer to preceeding messages. Linking is done by embedding a *stream identifier*, which is the result of a hash function applied to the encoded form of the message being referenced, into the the message containing the reference.

Transfer Message

The **transfer** message is sent by A to initiate a transfer of funds away from his account to another account. It is a function of

- the *account secret* a of A,
- the *account identifier* of B,
- the *generalized amount* vector,
- the recent state of the book of account A,
- and some random data r_A known only to A.

The value derived from the account history representing the recent state is also called the *ledger state* of that account.

The random data r_A is called the *transfer secret* of A and serves as commitment in the per transfer key exchange.

A must sign the message with the account secret a and the transfer secret r_A.

A must include the signed transfer message into her account history before sending it to B.

Accept Message

The second message, sent by B to A and called **accept** indicates whether B accepts the transfer or not. It is is a function of

- the first message,
- the generalized amount representing the part of the funds the B accepts to receive,
- the *account secret* of B,
- and some random data r_B known only to B.

To gracefully reject the transfer, B can send the confirmation message with an empty, zero amount vector.

B calculates the *shared transfer secret* $s = r_B R_A$ and derives the transfer signatue key k_S from it.

B must sign the message by (b, k_S) before sending it to A.

The accept message may include a ledger state but is not required to do so.

The accept message is not written to the book.

Confirmation Message

The confirmation message is sent by the sender A of funds to the recipient B of funds to indicate the commitment of the transfer process as given by the accept message. From the perspective of the funds transfer this is the last message that needs to be exchanged.

The message is a function of

- the second message,
- the secret random r_A
- and the *account secret* of A.

A can reject a confirmation, because of formal failure or any other reason.

A must include the confirmation message in his history if she accepts it.

A derives the shared transfer secret $s = r_A R_B$ and the transfer signing key k_S.

If the confirmation claims only a part of the transfer amount, A shall create a compensation message for the difference amount to the book.

Compensation Message

The sender of a transfer A can anytime cancel a previously initiated transfer by creating a related compensation message and writing it to his book.

Based on a partial confirmation, the sender can reimbourse the remaining funds to his account by creating a compensation message.

The compensation message may refer to a confirmation message and must be signed by (a, k_S) in that case.

Alternatively the compensation message may refer to a transfer message. It that case it must account for the whole amount of the transfer.

The compensation message is not transmitted.

Receipt Message

The receipt message is generated by the recipient of funds B after successful validation of a confirmation message. It refers to the confirmation message and is signed by (b, k_S).

The receipt message must be written to the book.

The receipt message is not transmitted to the transfer sender, in general.

6.6 Invoice Protocol

In the invoice scenario, the transfer is initiated by the recipient B by means of a *invoice* message. While this message does not affect the state of any book directly, it serves as a means to inform a buyer about the amount and transfer options. By further signing this message with the recipient secret and a random key, we can safely omit the transfer and accept messages in the protocol above. As a confirmation to the sender of the funds we require, in case of the invoice process, that the receipt message is transmitted to the sender of funds.

There is no need to give exact details on the invoice message in this context as it does not take active part in the protocol, in the sense of the definition above. A reference to the invoice message can however be afforded in the confirmation message envoyed by the buyer, thus linking the invoice information into the transfer process.

From practical considerations, the invoice message should include at least:

- Any information usually present.
- The account identifier of the invoice sender.
- One or more amount vectors representing possible payment options[6].

In this scenario, the receipt message is conveyed to the sender of funds A, leaving the option to embed additonal data, including things like transfer of loyality currency values, for example.

6.7 Acceptance Protocol

The acceptance protocol is an extension of the transfer protocol supporting the processing of conventional credit and debit cards as well as loyality schemes. It is based on a gateway capable of routing the card protocols into the scheme networks, possibly providing a two step mechanism, including an interactive authorisation step, resulting in a transfer message being sent to the merchants account and a second submission step that is executed when the merchant accepts the transfer. If the merchant rejects the transfer, the authorisation is reversed, respectively.

By embedding card transaction data into messages of the transfer protocol, the gateway can operate stateless using a single injection account to keep track of the transactions.

6.8 Exchange Protocol

While currencies are being considered unexchangable on the basic level of this construction, it is obviously possible to define a protocol to realise such an exchange between two parties, based on a agreement procedure and a shared transfer secret used to sign the usual accept-confirm stage.

As in the invoice case it seems impossible to generalize the method parties use to come to an agreement, but we can imagine a shared transfer secret being used to sign an *exchange-agreement* which is referred to by the transfer or accept.

[6]this is going to get very complex

7 Currencies

We have not lost any word about what currencies are in this construction. For a reason: it's not going to be defined, at least not in excess to the basic meaning of the word. The term *currency* in this context means anything countable or measurable that parties are willing to accept as a representation for value in their interactions.

Nevertheless, since currencies play a role in the generalized amount vector, a means to identify them is needed.

Everybody can create a currency by fabricating a *currency manifest* and publishing it. The manifest should be signed, but the method is not in scope. In first line, the manifest claims a name, more precisely a URI. It may contain whatever data the issuer wants to include.

If the URI used to identify the currency uses the http scheme, the issuer shall provide meaningful information about the currency under that URI.

Currencies are *not interchangeable* in the sense that one could be used for another, if present on the same account, for example. All kinds of values are considered distinct, and the conversion from one to another always requires at least two parties running some exchange protocol.

Any calculations related to conversion of values from one currency to another are not in the scope of this construction, but can be realised to any extent by the exchange protocol or extensions of it.

The recipient of a transfer is responsible for checking both the account and the currency before he accepts funds.

Many of the properties described in section 5 can be configured on a per currency basis. The currency manifest describes the rules for handling a particular currency.

8 Injection

The motivation of principle 6 is to clearly separate accounts that serve the purpose of injecting external values from accounts used for other operations. Each currency shall provide a whitelist indicating the acceptable source accounts, which shall have a balance less or equal zero. In terms of the balance vector this means that all components have negative sign.

Without injectors, a currency is empty and all transfers must be unfaithful because every account started with a zero balance and there is no other way to inject funds of that currency.

A currency gains representation when injector accounts are issueing transfer messages that are accepted by other accounts, increasing the negative balance of the injector accounts. Every amount transferred in that currency can be traced back to a subset of all injector transactions.

As stated, the word currency is used in a very basic meaning here, and so it can be perfectly possible to use a thing like a real estate as currency. The only thing needed is to have some unique and realiable identification of that estate and make an URI out of it. So in case of the real estate the manifest would also include the parcel number and there would be exactly one unit of that currency injected, representing that estate, by its legal owner. Subsequently, the owner could devide the estate into fractions, for example, in exchange to other currencies.

Reference Definitions

As part of this publication we want to provide definitions used in our implementation. These definitions are based Diffie-Hellman type cryptography over elliptic curves.

We use additive notation, small letters for private values and capital letters for curve points. Let G be the base point of some selected curve, then any account identifier A is related to the account secret a as

$$A = aG$$

Further we denote by $\lambda(Q) = \lambda Q$, where Q is a curve point, the value derived from Q to be used in exponent calculations. Typically, the x-coordinate modulo the the group order is taken for this[7].

9 Ledger State Calculation

An account history or book is a collection of messages $\mathbf{T} = \{t \mid t \in \mathbf{P}\}$ where $\mathbf{P}$ is the set of messages that are acceptable by the selected protocols. Such messages may be required to include a verification value, the *ledger state*, derived from the (ordered) contents of the book before the message was generated.

The ledger state is a function of $\mathbf{T}$, the *account secret* and the *transfer secrets*. Because signatures are used for history validation, it is not necessary to consider the secret values explicitly and we write $L(\mathbf{T})$ to denote the value of the ledger state for some history, omitting the account reference.

If we have two different states of the same account $\mathbf{T}_1$ and $\mathbf{T}_2$ where $\mathbf{T}_1 \subset \mathbf{T}_2$ we can faithfully expect that there is a set of messages $\Delta\mathbf{T} \subset \mathbf{P}$ so that

$$\mathbf{T}_1 \cup \Delta\mathbf{T} = \mathbf{T}_2$$

and equivalently for the ledger state

$$L(\mathbf{T}_1) \oplus L(\Delta\mathbf{T}) = L(\mathbf{T}_2).$$

As a consequence we can calculate the ledger state for every single message in respect to an account and simply sum up the states as messages are processed:

$$L(\mathbf{T}_{k+1}) = L(\mathbf{T}_k) \oplus L(t_{k+1})$$

The question whether the order of messages matters or not is answered by the choice of the $\oplus$ operation. If the operation is commutative, the order does not matter, otherwise messages must and can only be added up following the history one by one.

For the following discussion we shall assume that $L_0 = L(t_0)$ is the ledger state after the first message, the manifest. At any time at some physical location there is a most recent ledger state L_r calculated from the messages available at that location. L_r is embedded into protocol messages to prove their position in the account history.

The construction can operate accounts at distinct, possibly temporary separated locations, yet principle 6 remains effective, in that at any time the balance must not have crossed zero.

[7] must be checked for zero!

9.1 Commutative Ledger

This method of calculating the ledger state employes the fact that every message stored in the book is signed by the account secret and that this signature, in the elliptic curve case, consists of a number and a point. The ledger state is then defined, according to the conclusions above, as the sum of all the points included in the signatures of the messages processed so far.

The ECDSA signature of some data m, here represented by the result of a hash function $z = h(m)$, is, given private key a and random value k as the pair

$$(s, K) = (k^{-1}(z + a \cdot \lambda kG), kG)$$

While typically only $r = \lambda K$ is transmitted as part of the signature, we are free to use K as the value to be added to L_r to calculate the next most recent ledger state as $L_{r+1} = L_r + K$.

The advantage of this approach is that the ledger can be inquired selectively, giving the questioned party the opportunity to mask information not requested by replacing such records by the sum of their K's.

The risks and security properties of this approach have not been investigated.

9.2 Serializable Ledger

This method of calculating the ledger state is based on a simpler construction, namely $L_{r+1} = z = h(m)$, where L_r is embedded into m.

We use it with the restriction that every record in the book except the manifest must refer to exactly one ledger state and that two distinct records do not refer to the same ledger state.

The history of the book is then the sequence of all records as they are chained together.

10 Message Repertoire

The syntax definitions presented here show how messages are built-up in the reference implementation. All messages share a common format, namely that of a DER encoded `SEQUENCE` structure. The content of this structure is not defined formally, but follows some basic rules:

- If the first element of the sequence is a primitive `OBJECT IDENTIFIER` or `UTF8String`, that value should be used to determine the type of message. Not used here.
- If the first element of the sequence has an tag in the application class, the message type is given by the tag, according the application implied by the circumstances.
- The first element is also called the *body* of the message in case it is a constructed element.
- Any number of *attachments* can appear after the first element. The meaning, purpose and requirements for attachments are defined by the protocol.

Tag values are omitted, please refer to the source for that.

10.1 Common Data Elements

GeneralizedAmount

This structure is used to specify an amount as the superposition of values in certain currencies. It is represented by a map

```
GeneralizedAmount ::= SET OF SEQUENCE {
   currency Uri,
   numerator PositiveInteger,
   denominator PositiveNonZeroInteger OPTIONAL }
```

As for other places where this might apply we expect that protocol implementations handle messages in a way that preserves the order of collections embedded in the messages.

The denominator is optional and defaults to one.

AccountIdentifier

The account identifier shall, for practical reasons, be a suitable encoded form of the public key of the account. We use cryptography parameter agnostic opaque values containing the raw, compressed curve point.

```
AccountIdentifier ::= ECPublicKey ::= OCTET STRING -- compressed curve point
```

Signatures

The primary signature is calculated over the encoded form of the message body, compressed by hash function, according to ECDSA and added as the first attachment, usually.

This can be a standard ECDSA signature calculated by the account secret, in which case the protocol does not offer protection against replay attacks[8].

Alternatively, the signature can be calulated using a modified formula additionally involving the shared transfer secret. Given $r = \lambda q G$, calculate

[8]this might not be a problem, if all messages are kept along with the ledger

$$s = q^{-1}(z + ar + k_S)$$

where k_S is the transfer signing key derived from the shared secret as desribed above. This offers protection against replay attacks as it provides a freshness guarantee on all protocol messages except for the initial one.

We use the term *account signature* to refer to the case when the signature is calculate with the account secret only, and the term *transfer signature* for the case when a transfer secret is established and involved in the signature calculation process. The point is that the transfer signature is somewhat better in that it protects from replay attacks using such signed messages but it comes with an additional cost.

10.2 Transfer Message

The transfer message is the first piece of data exchanged in the sender initiated transfer protocol:

```
Transfer ::= SEQUENCE {
   body SEQUENCE {
      sender AccountIdentifier,
      recipient AccountIdentifier,
      ledgerstates SET OF OCTET STRING,
      amount GeneralizedAmount,
      r1 PublicKey OPTIONAL,
      timestamp TimeStamp OPTIONAL,
      expiration UTCTime OPTIONAL,
      ... },
   signature AccountSignature }
```

The `ledgerstate` field contains L_k for t_{k+1} in case of the serializable ledger.

The `signature` is calculated using the sender account secret and the data elements of the message.

Curve point `r1` represents the public part of the ephemeral shared transfer secret.

10.3 Accept Message

An accept messsage is always generated in response to the initial transfer message, to which it includes a reference. It shall also include the recipient public key for the purpose of efficient protocol operation to allow signature validation before reference lookup.

```
Accept ::= SEQUENCE {
   body SEQUENCE {
      recipient AccountIdentifier, -- signer of the accept
      transferReference Reference,
      amountAccepted GeneralizedAmount OPTIONAL,
      r2 PublicKey OPTIONAL,
      timestamp TimeStamp OPTIONAL,
      ... },
   signature AccountOrTransferSignature }
```

and sends it to the sender account.

To reject a transfer, the recipient generates a accept message with zero amount without incorporating it into the history. Following the principle of minimal overhead, the zero amount can be indicated by the absence of the `amountAccepted` field in the accept message.

10.4 Confirmation Message

The confirm message is used in two protocols: after the receipt of an accept message or after commiting to an invoice. It indicates that the sender, who includes it into his book, has finally given away the funds. The receiver subsequently generates a receipt message based on the confirmation and stores it in her book to indicate the receipt of the funds.

```
Confirm ::= SEQUENCE {
   body SEQUENCE {
      transferReference Reference OPTIONAL,
      acceptReference Reference OPTIONAL,
      externalReference Reference OPTIONAL,
      amountConfirmed GeneralizedAmount,
      timestamp TimeStamp OPTIONAL,
      ... },
   signature AccountOrTransferSignature }
```

10.5 Receipt Message

The recipient of a confirm message generates the receipt message and writes it to his book.

```
Receipt ::= SEQUENCE {
   body SEQUENCE {
      transferReference Reference OPTIONAL,
      confirmReference Reference OPTIONAL,
      amountReceived GeneralizedAmount,
      timestamp TimeStamp OPTIONAL,
      ... },
   signature AccountOrTransferSignature }
```

10.6 Error Message

The error message shall be used to indicate the receipt of a malformed or invalid message to the sender of that message.

Example Flows

This section shall briefly explain some of the basic flows in the exchange protocol as far as desribed before.

11 Ordinary Transfer

The flow of message for a transfer from sender Alice to recipient Bob. Messages are always booked by their originators.

Message	Origin	Amount	Booked?	Sent?
Transfer	Alice	$\mathbf{v}_0$	✓	✓
Accept	Bob	$\mathbf{v}_0$	✓	✓
Confirm	Alice	$\mathbf{v}_\omega$	✓	✓
Receipt	Bob	$\mathbf{v}_\omega$	✓	

Typically $\mathbf{v}_\omega \leq \mathbf{v}_0$ is assumed, but parties may agree to anything else.

12 Cancelled Transfer

Cancellation does not involve any other party then the initiator of the transfer.

Message	Origin	Amount	Booked?	Sent?
Transfer	Alice	$\mathbf{v}_0$	✓	✓
Compensate	Alice	$\mathbf{v}_0$	✓	

The compensation must contain the full transfer amount.

A cancelled (i.e. fully compensated) transfer must not allow any subsequent accept message.

13 Partial Accept

This flow is for the case the recipient wants to receive only part of the funds.

Message	Origin	Amount	Booked?	Sent?
Transfer	Alice	$\mathbf{v}_0$	✓	✓
Accept	Bob	$\mathbf{v}_1$		✓
Confirm	Alice	$\mathbf{v}_1$	✓	✓
Compensate	Alice	$\mathbf{v}_0 - \mathbf{v}_1$	✓	
Receipt	Bob	$\mathbf{v}_1$	✓	

14 Rejected Transfer

In this flow the recipient does not accept any of the transferred funds.

Message	Origin	Amount	Booked?	Sent?
Transfer	Alice	$\mathbf{v}_0$	✓	✓
Accept, Error	Bob	0		✓
Compensate	Alice	$\mathbf{v}_0$	✓	

15 Invoice Processing

Invoice processing can omit the transfer and accept messages; the latter is replaced by an invoice message containing the account identifier of the receiving account.

While the invoice message is not booked, the related confirm message must contain a reference to the unmodified invoice document.

15.1 Basic Flow

In the basic flow the amount remains unchanged during the flow.

Message	Origin	Amount	Booked?	Sent?
invoice	Bob	$\mathbf{v}_0$		✓
Confirm	Alice	$\mathbf{v}_0$	✓	✓
Receipt	Bob	$\mathbf{v}_0$	✓	✓

15.2 TIP Flow

In the TIP flow, the sender of funds may add funds before confirming.

Message	Origin	Amount	Booked?	Sent?
invoice	Bob	$\mathbf{v}_0$		✓
Confirm	Alice	$\mathbf{v}_0 + \mathbf{v}_t$	✓	✓
Receipt	Bob	$\mathbf{v}_0 + \mathbf{v}_t$	✓	✓