Towards a Generalized Payment Model for Internet Services[⋆]

Michael Fischer,[1] Harald Gall,[1] and Manfred Hauswirth[2]

[1] Distributed Systems Group
Technical University of Vienna,
A-1040 Vienna, Austria
{fischer,gall}@infosys.tuwien.ac.at
[2] Distributed Information Systems Lab, EPFL
1015 Lausanne, Switzerland
manfred.hauswirth@epfl.ch

Abstract. Prerequisite for the success of new business models in the Internet, such as pay-per-view, will be an efficient and interoperable electronic payment system. Many protocols and frameworks for various business domains exist. However, they are mostly incompatible which makes it hard for service providers to design for change. We investigated several standard payment scenarios and configurations and analyzed shortcomings of existing payment schemes. As a result, we developed expressive, common payment abstractions and came up with a generalized payment model which hides the payment mechanisms used, but offers a common, high-level interface and supports a wide range of business models. In this paper, we present our generalized payment model and its accompanying security model for Internet services. We discuss its abstractions and protocols and evaluate it in an Internet-scale push system.

1 Introduction

E-commerce sites which require some form of payment for part of their services are the main segment of growth in the Internet landscape. At the moment the standard payment instrument are credit cards: When the customer decides to buy, s/he fills in a form with her/his credit card information which is then transmitted to the vendor's site via the *Secure Socket Layer* (SSL) protocol (HTTPS) [7].

Only few sites offer electronic payment via a payment protocol such as *Secure Electronic Transaction* (SET) [26]. Even fewer sites employ micro-payment protocols such as Millicent [10]. However, electronic payment is important for e-commerce applications because it enables highly relevant business models such as pay-per-view, volume-based, time-based, pre-paid, or post-paid.

[⋆] This work was supported in part by the European Commission under contract IST-1999-10288, project OPELIX (Open Personalized Electronic Information Commerce System).

Assuming that electronic payment will be employed on a large scale in the near future and on the basis of the state-of-the-art in payment systems, a critical problem emerges: There will be a number of competing standards which are incompatible and require applications to be tightly coupled with them. This impedes component-oriented design for change and makes it hard for customers to interact with vendors who use different payment protocols. Additionally, vendors (or customers) may want to delegate payment handling to specialized third-parties who offer appropriate security and sufficient proofs of their transactions. Such payment intermediaries would have to support a wide range of different payment protocols and business models at reasonable costs to make a revenue out of their business.

The analysis of current payment protocols shows that most of the protocols are broker-centric, i.e., the broker is involved in the protocol during the payment operation. For example, NetBill [28] requires a server for billing and dispute handling, DigiCash [3] needs one to detect double-spending, and SET [26] must validate credit card data at a server. One exception is the Millicent [10] protocol in which the integrity of the electronic "currency" (scrip in Millicent terminology), can be validated by the vendor. A possibly disadvantageous property of some protocols is that they define the payment and delivery process as an integrated transaction, e.g., NetBill, which violates the component-based approach of software composition and restricts flexibility. A limiting factor in many micro-payment protocols is the requirement to subscribe to a vendor's payment broker to obtain the virtual currency.

Besides these problems the current payment approaches are useful in many settings but still do not meet the requirements of modular system design and do not adequately address flexible business models as in our push system case study: (1) we use the Minstrel [13,30] Internet-scale push systems with its optimized distribution infrastructure which requires decoupling of payment and delivery, i.e., an arbitrary number of delivery components may use a single payment component; (2) the business model and the associated payment model are not known a priori; (3) there is no interoperability between different payment schemes; and (4) the customer/vendor has little freedom in choosing the payment instrument/provider. These constraints motivated our approach.

In this paper we introduce our generalized payment model which supports account-based and token-based payment schemes. The security model inspired by the Millicent payment scheme has been extended to meet the requirements of the various business models we want to support in the Minstrel push system. We have paid much attention to keep the communication efficient and the transaction costs low. This makes the system well-suited for small-value transactions, so-called *micro-payments* [12], but also addresses a wide range of other business models.

This paper is organized as follows: Section 2 positions our approach in respect to related work. Section 3 characterizes common payment abstractions and introduces our generalized payment model. Section 4 discusses the security related aspects of our approach. In Section 5 we evaluate our model in connection with Minstrel and Section 6 concludes the paper.

2 Related Work

Many electronic payment systems rely on non-electronic payment systems or extend them. Often they use existing infrastructures such as banks or credit card companies and create an electronic communication system between vendor, customer, and bank (or credit card company). Besides this "traditional" approach as first realized by First Virtual Holdings [29] or SET [27] for online credit card payments, new electronic payment systems have been introduced [22,23,33]. These systems focus on reduction of transaction costs, improvement of transaction speed and/or support of anonymity of the customer. So far little attention has been paid to the issues of generalized payment frameworks and payment interoperability.

2.1 E-Commerce Systems

SEMPER [19] supports different payment instruments by implementing a *Generic Payment Service Framework* (GPSF) [25] which provides a well-defined interface for higher level services but still having enough flexibility in supporting different payment models, e.g., check-like or cash-like. But payment is only a small aspect in SEMPER since it has been designed more generally to provide security for electronic commerce on the Internet.

2.2 Payment Frameworks

One critical aspect in payment instruments is their integration into e-commerce applications. Beyond other aspects, it is required to address problems such as security, different operating system platforms, programming language technologies, design of a common *Application Programming Interface* (API), peculiarities of the payment instrument, and the underlying protocol. At least, the latter two topics have to be addressed by payment frameworks to facilitate generic application development. The *Universal Payment Application Interface* (U-PAI) [18] proposes a standard interface to multiple payment mechanisms. It does not address negotiation for parameters before a payment transaction begins; nor does it explicitly address issues like refunds. Additionally a distributed object infrastructure such as CORBA [31] is assumed or at least *Remote Procedure Calls* (RPC) for non object-oriented applications. Further, a clear security and trust model are not provided [1].

The *Simple Wallet Architecture for Payments, Exchanges, Refunds, and Other Operations* (SWAPEROO) [21] is part of the Stanford Digital Libraries project and tries to tackle some of the open issues of U-PAI, for example, instrument and protocol negotiation. The proposed architecture has been used to implement a digital wallet on the PalmPilot [1].

2.3 Payment Selection Protocol

Payment selection tries to tackle the problem of selecting a payment method by negotiating the appropriate payment instrument and transport protocol. An additional goal is to provide methods for encapsulating payment instrument messages for transport in various application environments, especially the World Wide Web and e-mail.

The goal of the *Joint Electronic Payments Initiative* (JEPI) [32] was the development of an Internet payment negotiation protocol as an extension to HTTP which should enable automated payment negotiation between computers. This initiative has been redeemed in 1996 by several other activities of the W3C in the e-commerce area.

2.4 Payment Protocols

More than 50 payment protocols [33] were proposed within the last decade. None of them has reached the critical mass to be successful and only a handful are still supported by banks and merchants.

Some protocols require chip cards which hold cryptographic information, e.g., tokens which represent coins or keys required for message signing. Chip cards, popularly referred to as *smart cards*, are one of the most important secure hardware devices [22].

Account-Based Systems. CyberCash – as a representative for this type of systems – was started in April 1995 but has stopped the online-payment system in Germany [6] by the end of 2000 [15] due to low interest of users. The reason why CyberCash is listed here is the fact that the system does not really store coins in the customer wallet: CyberCash provides a safe passage over the Internet for credit card transaction data. It takes the data that is sent to the CyberCash server by the merchant, the server resides between the merchant and the acquiring bank, and passes it to the merchant's bank for processing. The acquiring bank processes the credit card transaction as they would process transactions received through a *point of sale* (POS) terminal in a retail store [20].

Token-Based or Cash-Like Systems. Token-based systems try to map the concept of real money onto cryptographic properties. MicroMint and Millicent are two examples for such systems using different concepts for their virtual currency.

The MicroMint [24] micro-payment system is based on the concept of k-way hash function collisions. A "coin" is represented by sets of values $(w_0, w_1, ..., w_k)$, where $h(w_0) = h(w_1) = ... = h(w_k)$. A k-way hash function means that k different input values map to the same output value. Such coins, worth some cents, are produced by a broker who sells them to users. The customers pay at a vendor with these coins. The vendor exchanges these coins with "real" money at the broker. A successful implementation of this scheme in Java for purchasing

Web pages has been done by our group [14]. MicroMint has the disadvantage of requiring a large hardware base for minting coins, which is infeasible for small service providers.

Millicent [5,10] is a lightweight protocol for electronic commerce over the Internet which enables efficient pay-per-view business models via HTTP [17]. It uses a form of electronic currency (scrip) which is intended for small value transactions, ranging from a minimum of one cent or less to a maximum of approximately 5 Euro.

3 Towards A Generalized Payment Model

Payment models classify the digital payment systems according to the necessary flow of information between the participants of an electronic transaction [1]. The distinguishing property is the presence or absence of a direct communication between the customer and the vendor. In *direct cash-like systems* the customer withdraws money from the payment server, hands the payment tokens over to the vendor who in turn deposits the payment with its payment broker [33]. *Direct account-based systems* resemble conventional cheque systems. They allow the customer to hand over a payment authorization to the vendor, who presents the payment authorization to its payment broker and who in turn redeems it from the payment server [33]. In the case of indirect payment models, *indirect push* and *indirect pull*, the communication of the payment itself concerns only the initiator of the payment, which may be the customer or the vendor, and the payment server and payment broker. The receiver of the payment is only informed about the successful payment.

3.1 A Generalized Payment Model

For the support of Minstrel [13] with its efficient dissemination structure, a generalized payment model was required to facilitate the flexible implementation of business models such as pay-per-view, volume-based, time-based, pre-paid, or post-paid. Minstrel is a lightweight push system for information commerce using a highly efficient HTTP-based dissemination infrastructure consisting of information broadcasters and repeaters. Specific client software is used to subscribe to and receive from information channels provided by the broadcasters.

Figure 1 shows the pay-per-view scenario we use in Minstrel as a UML use case diagram. The scenario involves the following actors:

- The *vendor* sends *offers* about information it wants to sell to *customers*.
- On the basis of the received offers the customer can decide to request a *shipment*, e.g., a piece of the information such as news articles, software, images, etc.
- Depending on the business model the customer may have to pay before or after the shipment is sent or may just have to present a certificate (*receipt*) which acts as a proof of payment.

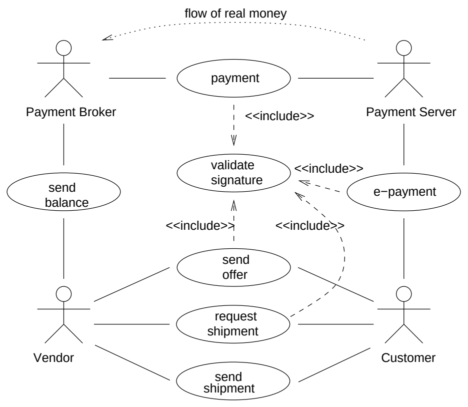

Fig. 1. Payment Use Case Diagram

- To perform a payment the customer contacts a payment server (defined by the vendor or the customer's favorite one).
- The payment server accepts certain kinds of electronic payment and issues signed receipts for successful payments.
- This server is operated by or cooperates with *payment brokers* (financial institutions that are responsible for the real-money transfer from the payment server's account to the vendor's account). We consider this not necessarily to be a single entity: It may be a bank, the customer's ISP who offers a charging service or any other business entity/consortium. The identity of all parties can be guaranteed through, for example, a *Public Key Infrastructure* (PKI).

The benefit for the vendors in this setting is that they do not have to take care of payment issues. However, it is not required to route all messages through a payment server [8] or proxy [2] since digital signatures ensure the proper operation of the protocol and can serve as evidence in case of arguments. The existing infrastructure of ISPs, banks, or credit card companies can be used to offer payment services to customers supporting their favorite payment scheme while guaranteeing payment to the vendor account through the certified identity of the payment server. Thus also the customer is freed from having the "correct" payment instrument installed: One payment instrument is sufficient to pay at the payment servers which can mediate between all kinds of payment instruments.

3.2 Mappings

We now describe how the direct cash-like and direct account-based models introduced at the beginning of this section can be mapped onto our generalized payment model:

Direct Cash-Like: The customer asks for "coins" at the preferred payment server and does a payment for the benefit of the vendor. The payment server in turn returns certificates, i.e., *receipts*, representing the requested monetary value, e.g., 1, 2, or 5 cents each. The customer may now obtain services from the vendor by handing over the required amount expressed by the receipts. The vendor then sends the receipts to the payment broker for redemption.

Direct Account-Based: The customer creates a "check" certificate by signing a guarantee statement from the payment server and the amount to pay to the vendor's account. The vendor validates the "check" and sends the payment authorization to the payment broker for redemption.

The other two models–indirect push and indirect pull–also can be mapped onto our model by creating appropriate certificates which are sent to the participating payment server and payment broker, respectively.

3.3 2+1+1 Party Generic Payment Protocol

This section takes a closer look at the sequence of actions during a payment interaction (for example, as performed in Minstrel) and describes the 2+1+1PGPP protocol for payment. Figure 2 shows the sequence diagram for payment in our model.

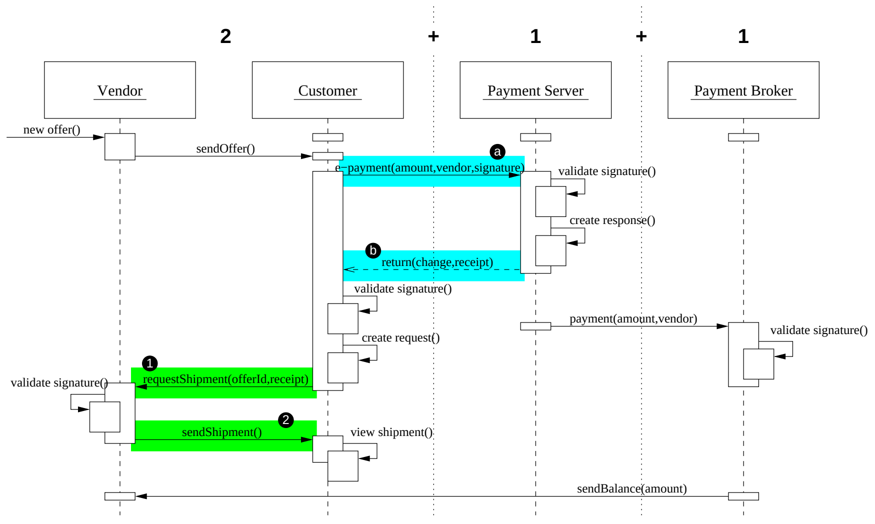

Fig. 2. Payment Sequence

Initially the vendor sends an offer with price and payment information to the customer (this is not part of the protocol). If the customer accepts the payment terms and wants to buy, s/he authorizes the payment and sends a payment request using the preferred instrument, i.e., the payment instrument the customer and the payment server agreed on, to the payment server ((a) in Figure 2). Besides the information required for the payment instrument the request includes vendor information and the customer digital signature. The payment server validates the request to check its authenticity, performs the payment according to the payment instrument, generates the *receipt*, and returns it to the customer upon successful completion ((b) in Figure 2). The receipt is generated by signing the customer-signature together with the amount and the vendor information (the security aspects will be discussed in detail in Section 4).

In a time-based business model, for example, this payment step may be skipped if the customer already has a valid receipt which may be presented to the vendor as a proof of payment as described below (Steps 1 and 2 in Figure 2). The customer validates the attribute certificate to proof the authenticity of the payment server and can now be sure that the server has accepted the payment and the amount will be transferred to the vendor's account. This transfer can be done via an arbitrary payment protocol and employs similar authentication checks using digital signatures. Upon successful validation the customer sends a shipment request to the vendor containing offer information, customer information and information from the certificate from the payment server's response (amount, id of payment server, serial number, expiration date, payment server signature) to the vendor (Step 1 in Figure 2). The vendor validates the shipment request by checking the payment server's signature which guarantees that the required amount will be paid to the vendor's account. Upon successful validation the vendor sends the shipment plus an updated version of the receipt to the customer (Step 2 in Figure 2). Steps 1 and 2 may be repeated as many times as the receipt guarantees a value that is higher than the requested shipment.

The payment server transfers the paid amount to the payment broker using an arbitrary protocol either immediately or when a certain threshold amount has been accumulated, or in regular intervals, or according to some other defined procedure they agreed upon.

The scenario of Figure 2 can be mapped directly to a pay-per-view business model but can also be generalized easily to other business models depending on the semantics assigned to the attribute certificate and the additional data it carries. For example, in a time-based business model the payment would happen once in a while and the attribute certificate would hold a timestamp until which the payment is valid; in a volume-based model the certificate would hold a counter.

4 Security Model for the 2+1+1PGPP

This section describes our approach to ensure the receipt's integrity and authenticity among the participants. Our approach is based on the technique described in [10] and uses shared secrets which must be negotiated once via a secure channel (e.g., SSL). The security model also encourages the use of receipts for business models on a pay-per-view basis in a more flexible way than most micro-payment schemes by allowing the payment server to re-issue receipts, e.g., subtraction of the amount due and returning the updated receipt to the customer (see Section 4.4 for details).

4.1 Abbreviations

The abbreviations given in Table 1 will be used in the subsequent discussion of the security mechanisms.

Abbreviation	Description
$S_{vc}c$	Signature generated by the customer and used in vendor–customer relationship.
$S_{pc}c$	Signature generated by the customer. This signature is used for messages exchanged between payment-server–customer.
$S_{pc}p$	Signature generated by the payment-server in the payment-server–customer relationship.
$S_{pv}p$	Signature generated by the payment-server to ensure message integrity between payment-server and vendor.
P_{ID}	Unique payment-server ID.
V_{ID}	Unique ID of the vendor.
$C_{ID}P$	Unique customer ID issued by the payment-server.
$C_{ID}V$	Unique customer ID issued by the vendor.
sk_{vc}	Shared key, i.e., secret, between vendor and customer.
sk_{pc}	Shared key between payment-server–customer.
sk_{pv}	Shared key between payment-server–vendor.
oid	Unique offer ID issued by the vendor.
rid	A unique ID for a receipt generated by the payment-server which allows un-ambiguous identification of a payment transaction.
$value$	Amount of monetary units to be transfered from the customer's account to the vendor's account.
t_xc	A timestamp generated by the customer and required to identify the key which has been used to sign a message at a particular time. A serial counter may be part or used instead of a date/time field as well.
t_xp	Timestamp generated by the payment-server.
$props_{vc}$	A canonical representation of vendor–customer properties (e.g., textual information describing an offered item, liability issues, etc.).
$props_{pv}$	payment-server–vendor properties (e.g. expiration date of the receipt, etc.)

Table 1: Abbreviations used in formulas

4.2 Basic Setting

Message security is achieved by sending the message combined with its signature. This signature is calculated by applying a secure hash function (e.g., SHA or

MD5) to the message and the associated secret, yielding the following *Protocol Data Unit*: $PDU = message + hash(message + secret)$. The addressee can verify the signature by recalculating the signature using the shared secret. The security stems from the fact that it is infeasible for an outside observer to generate the signature without knowledge of the secret or deduce the secret by knowing the message and the signature. In the subsequent discussions of the payment process we assume the following setting (without constraining generality):

1. The customer is already registered at the vendor's push system (this implies that sk_{vc} for message authentication has been exchanged, the customer has a valid C_{ID}, and knows V_{ID}).
2. The customer is registered at a payment-server, i.e., the customer and the payment-server have agreed on sk_{pc} and the account's balance is sufficient for payment.
3. The customer has received an offer from the vendor containing an *oid* and a *value*.
4. The vendor and the payment-server have already authenticated themselves and agreed on sk_{pv}.

4.3 Establishing end-to-end Security

To pay for an offer the customer creates a unique fingerprint for this payment transaction by performing the following operation which yields the vendor–customer signature:

$$S_{vc}c = hash(t_xc + C_{ID}V + V_{ID} + oid + sk_{vc}) \ . \tag{1}$$

Additional information describing the offer, e.g., a URL or business conditions, may be included in the signature via the optional $props_{VC}$ properties field. The payment model does not make any assumptions on how the data has to be included during the signature generation process. Since it is conceivable to use a secure hash sum calculated by the customer instead of inserting the properties data itself, the preferred scheme is *blind signature* [4] and transmission of data to the payment-server can be omitted. Then the customer constructs a signed payment request (some payment protocols do this implicitly, e.g., Millicent)

$$S_{pc}c = hash(t_xc + C_{ID}P + V_{ID} + value + S_{vc}c + sk_{pc}) \ . \tag{2}$$

and sends the request, i.e., t_xc, which is required here for replay detection, $C_{ID}P$, V_{ID}, $value$, $S_{vc}c$, some optional information in the $props_{pv}$ field and $S_{pc}c$, along with the payment protocol information, e.g., specific information from SET [27] or other protocols, to the payment-server.

Upon successful completion of the payment transaction the payment-server returns a receipt to the customer holding two signatures:

$$S_{pv}p = hash(t_xp + P_{ID} + value + rid + S_{vc}c + sk_{pv}) \ . \tag{3}$$

$$S_{pc}p = hash(t_xp + P_{ID} + value + rid + S_{pv}p + sk_{pc}) \ . \tag{4}$$

The first signature is the proof for the vendor that the payment is authentic and will be fulfilled by the payment-server. The second signature is for the customer that the payment-server has accepted the payment request.

The payment-server will return the following data items to the customer: t_xp the timestamp when the receipt was generated, i.e., signed, rid the unique ID for this receipt, $props_{pv}$ some optional information from the payment-server to the vendor, $S_{pv}p$ the signature for the vendor, $props_pc$ some optional information for the customer (e.g., accounting information), and sk_{pc} the signature for the customer by the payment-server.

It is feasible to use *Public Key Cryptography* (PKC) for the signature generation process. The appealing thing of the approach using PKC is, that only a single signature by the payment-server is required and the validation procedure of the receipt for customer and vendor is the same. Disadvantageous on PKC is the requirement for about 10^3 times more computational power to fulfill cryptographic operations in comparison with cryptography using shared secrets.

Now the customer validates the payment-server–customer signature $S_{pc}p$ by recalculating the signature using the shared secret. If the validation of $S_{pc}p$ succeeds, the customer can assume that $S_{pv}p$ is correct too. Having successfully paid, the customer requests a shipment by "reusing" information artefacts from the previous payment process and sending C_{ID}, oid, some information from the receipt (t_xp, P_{ID}, $value$, $S_{pv}p$), and $S_{vc}c$ to the vendor.

The vendor uses this information and the associations between secrets and signatures to validate the integrity and authenticity of the request. First the vendor recalculates the customer's signature by inferring the shared secret sk_{vc} using the customer ID $C_{ID}V$ and then recalculating the signature $S_{vc}c$. Then the vendor uses the P_{ID} from the request to infer sk_{pv} which is used to recalculate $S_{pv}p$. If the comparison succeeds the payment (receipt) is valid, which means that the following condition holds:

$$S_{pv}p = hash(data_{pv} + hash(data_{vc} + sk_{pc}) + sk_{pv}) \ . \tag{5}$$

i.e., the payment-server's signature of the receipt for the vendor to validate a customer request, where $data_{pv}$ denotes the canonical representation of payment-server–vendor properties and $data_{vc}$ denotes the canonical representation of vendor–customer properties (t_xc, rid, oid, etc.). The above process ensures end–to–end security in terms of authenticity and non-repudiation between payment-server and vendor by the use of shared secrets. The described method is sufficient for use within the Minstrel push system, since it is targeted on efficient delivery of low-value items by assuming a long term customer–vendor relationship where anonymity is not important. For higher security requirements it may be enhanced by using asymmetric cryptography in the signature generation/validation process to achieve privacy or to generate legal evidences.

4.4 Receipt Rewriting

Receipt rewriting is the alteration process of a receipt's components by the receipt-issuing authority. Receipt rewriting through the vendor is enabled by the use of a shared key between payment-server and vendor and allows a vendor to behave as a payment-server, i.e., to alter the content of a receipt and to return a new signed receipt to the customer. This activity correlates with the process of reducing the value of scrip in Millicent. Dependent on the business model, this feature is used by the vendor to reduce the guarantee amount of a receipt by the price of a particular service and to return the rewritten receipt to the customer as change. The change in turn can be used by the customer to pay for subsequent service requests offered by that vendor. This facilitates the implementation of an efficient payment scheme, since only one request/response pair is required for the service request, payment and delivery.

In the case of rewriting a receipt, the vendor needs to update the payment-server's timestamp t_xp, serial counter t_xc (see Equations 1 and 2) and *value* field of the receipt. The vendor also needs to update its receipt database to reflect the modifications to the t_xp and t_xc field. Reusing the data from the shipment request validation process is possible, when assumptions about the initial value of the serial field of the customer timestamp t_xc can be made, e.g., if it can be assumed that the value is 0. Otherwise the signature has to be regenerated with the actual value of t_xc. Then the signatures, i.e., the payment-server–vendor signature and payment-server–customer signature (see Equations 3 and 4), are regenerated and the updated receipt items and signatures are sent back to the customer along with the information from the original request.

On receiving the modified receipt data, the customer increments the serial values in the t_xc and t_xp fields, validates the vendor's signature and writes the modified data back to the stored receipt.

To issue a new shipment request, the customer generates a new signature to maintain the security properties. This is achieved by using the updated timestamp t_xc to regenerate the $S_{vc}c$ signature value.

4.5 Unused Receipts

Another nice property about receipt rewriting is the possibility of returning receipts with an unused or partially used amount to the payment-server. Clearly, this cannot happen ad hoc by the customer, since the state of a receipt (used/unused) can only be determined by the vendor. Consequently, the vendor's authorization is required, e.g., by setting the t_xp field of the receipt to indefinite and thus invalidating the receipt, to return the receipt for redemption.

5 Validation

The validation of our approach has been done with the Minstrel push system. To perform this validation, a payment framework, the Minstrel Payment Framework (see Figure 3), has been created which comprises the following components:

– A 2+1+1PGPP plugin for the vendor (broadcaster) to accept receipts received from the customers.

– The client (receiver) wallet to handle payments and receipts.
– A payment server to accept payments from the customers and issue receipts.
– A payment broker for issuing currency. In our setup we used Millicent for this purpose.

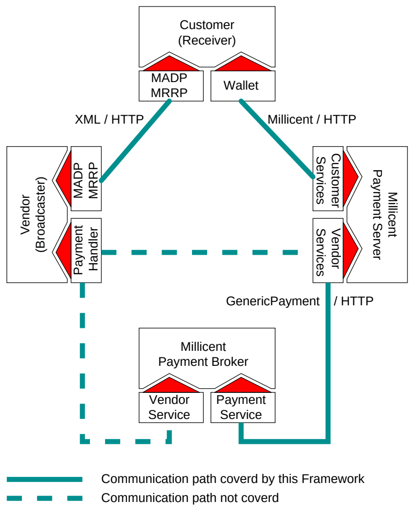

Fig. 3. Overall Architecture of the Payment Framework

The communication between the different entities in the network is based on XML message exchanges via HTTP and Java Servlets [16]. On-the-wire protocols between vendor and client are the *Minstrel Active Distribution Protocol* (MADP), which is used to actively disseminate information to the client, and the *Minstrel Receiver Request Protocol* (MRRP), which is used by the client to request information from the vendor [11]. The remaining communication is based on a generic method invocation mechanism on top of HTTP.

If a customer requests information in our pay-per-view business model, the payment information, i.e., the receipt obtained for a valid payment from the payment server, is piggy-backed onto with a receiver request (MRRP) from the customer to the server. The above payment is based on the offer information defined by the business model the user has received via MADP from the vendor before.

Design and implementation of the payment framework together with the payment model are discussed in detail in [9].

6 Conclusions

In this paper we have presented a generalized payment model for Internet-based services. We have described the use cases and the interactions which have to take place independent of the payment instrument used, and the artifacts and roles involved. Our approach decouples payment from applications in a way which enables the exchange of payment components and supports component-orientation also for payment. The 2+1+1PGPP payment approach supports interaction with arbitrary payment systems and facilitates the implementation of different business models via the concept of extensible receipts. It also offers "off-line" operation and requires only minimal resources at the vendor. Its component-oriented approach naturally supports distribution of the components in a network which offers new business opportunities for specialized payment services and frees both vendors and customers from many problems in the domain of e-payment. Since our approach introduces new security concerns we have presented feasible solutions for that as well. 2+1+1PGPP was validated with a pay-per-view business model case study in the Minstrel push system.

Though the technical foundations for security and payment exist to a large degree, there is still a lack of organizational ones. The X.509 *Public Key Infrastructure* (PKI) and its deployment is a good example on how long it can take till a new technology becomes accepted by the user community.

More research work has to be done in the area of a unified architecture for micro-payment systems and the development of organizational foundations for an efficient payment and clearing infrastructure.

References

1. J. L. Abad Peiro, N. Asokan, M. Steiner, and M. Waidner. Designing a generic payment service. *IBM Systems Journal*, 37(1):72–88, 1998. http://www.semper.org/info/212ZR055.ps.gz.
2. ALLCASH. Das Micropayment-System von ALLCASH, 2002. http://www.allcash.de/.
3. D. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.
4. D. Chaum. Blind Signatures for Untraceable Payments. In D. Chaum, R.L. Rivest, and A.T. Sherman, editors, *Advances in Cryptology Proceedings of Crypto 82*, pages 199–203, 1982.
5. Compaq Computer Corporation. Millicent Microcommerce Network, 1997. http://www.millicent.com/home.html.
6. CyberCash GmbH. CyberCash GmbH, 1997. http://www.cybercash.de/.
7. T. Dierks and C. Allen. The TLS Protocol Version 1.0, 1999. RFC 2246. http://www.ietf.org/rfc/rfc2246.txt.
8. Firstgate click&buy. Das neue Zahlungssystem im Internet, 2002. http://www.firstgate.de/.
9. Michael Fischer. Towards a Generalized Payment Model for Internet Services. Master's thesis, Distributed Systems Group of the Information Systems Institute, Technical University of Vienna, September 2002. http://www.infosys.tuwien.ac.at/Research/Masters.html.
10. S. Glassman, M. Manasse, M. Abadi, P. Gauthier, and P. Sobalvarro. The Millicent Protocol for Inexpensive Electronic Commerce. In *Fourth International World Wide Web Conference, Boston, Massachusetts, USA*. O'Reilly, Nov. 1995. http://www.w3.org/Conferences/WWW4/Papers/246/.

11. Stefan Haberl. An efficient and open implementation of the Minstrel broadcasting infrastructure. Master's thesis, Distributed Systems Group of the Information Systems Institute, Technical University of Vienna, 2000. http://www.infosys.tuwien.ac.at/Teaching/Finished/MastersTheses/Haberl/haberl.pdf.zip.
12. Vesna Hassler. *Security Fundamentals for E-Commerce*. Computer Security. Artech House Publishers, 2001.
13. Manfred Hauswirth. *Internet-Scale Push Systems for Information Distribution-Architecture, Components, and Communication*. PhD thesis, TU Vienna, 1999. http://www.infosys.tuwien.ac.at/Staff/pooh/diss/Thesis.ps.
14. Manfred Hauswirth, Vessna Hassler, Michael Fischer, and Rober Bihlmeyer. MiMi: a Java implementation of the MicroMint scheme. In *Proceedings of the 2nd World Conference of the WWW, Internet, and Intranet (WebNet '97), Toronto, Canada*, November 1997. http://www.infosys.tuwien.ac.at/reports/repository/TUV-1841-97-04.ps.
15. heise online. CyberCash: Aus für elektronisches Geld, 2000. http://www.heise.de/newsticker/data/ad-19.12.00-000/.
16. Jason Hunter and William Crawford. *Java Servlet Programming*. O'Reilly, 2001.
17. IETF. *Hypertext Transfer Protocol – HTTP/1.0*, 1996. http://www.ietf.org/rfc/rfc1945.txt.
18. Steven Ketchpel, Hector Garcia-Molina, Andreas Paepcke, Scott Hassan, and Steve Cousins. UPAI: A Universal Payment Application Interface. In *USENIX 2nd e-commerce workshop*, 1996. http://citeseer.nj.nec.com/ketchpel96upai.html.
19. G. Lacoste, B. Pfitzmann, M. Steiner, and M. Waidner, editors. *SEMPER – Secure Electronic Marketplace for Europe*, volume 1854 of *LNCS*. SV., 2000.
20. Keith Lamond. Credit Card Transactions: CyberCash, 1996. http://www.virtualschool.edu/mon/ElectronicProperty/klamond/Cyberpmt.htm.
21. N. Daswani and D. Boneh and H. Garcia-Molina and S. Ketchpel and A. Paepcke. SWAPEROO: A Simple Wallet Architecture for Payments, Exchanges, Refunds, and Other Operations. In *Proceedings of the Third USENIX Workshop on Electronic Commerce*, 1998. http://citeseer.nj.nec.com/daswani98swaperoo.html.
22. Donal O'Mahony, Michael Peirce, and Hitesh Tewari. *Electronic Payment Systems*. Artech House Computer Science Library, June 1997.
23. Michael Peirce. Payment mechanisms designed for the Internet, 2001. http://ganges.cs.tcd.ie/mepeirce/Project/oninternet.html.
24. R. L. Rivest and A. Shamir. PayWord and MicroMint: Two simple micropayment schemes. 1996.
25. SEMPER. Advanced Services, Architecture and Design, 1999. http://www.semper.org/info/.
26. SET Secure Electronic Transaction LCC. *SET Secure Electronic Transaction Specification – Book 3: Formal Protocol Definition*, 1997. Version 1.0. http://www.setco.org/download/set_bk3.pdf.
27. SET Secure Electronic Transaction LLC. SET Secure Electronic Transaction Specification, 1997. http://www.setco.org/.
28. M. Sirbu, B. Cox, and J. D. Tygar. NetBill Security and Transaction Protocol. pages 77–88, 1995. http://www.ini.cmu.edu/NETBILL/pubs/Usenix.html.
29. Stein, Stefferud, Borenstein, and Rose. The Green Commerce Model, May 1995. Memo.
30. The Minstrel Development Team. Minstrel Internet-Scale Push System, 1997. http://www.infosys.tuwien.ac.at/Minstrel/.
31. S. Vinoski. Distributed Object Computing With CORBA. *C++ Report*, 7/8, 1993.

32. W3C. Joint Electronic Payment Initiative, 1995. http://www.w3.org/ECommerce/JEPI.html.
33. R. Weber. Chablis - Market Analysis of Digital Payment Systems, Aug. 1998. http://chablis.informatik.tu-muenchen.de/MStudy/.