LAYERED ARCHITECTURE FOR SECURE E-COMMERCE APPLICATIONS

Amir Herzberg and Igal Yoffe
Computer Science Department,Bar Ilan University
Ramat-Gan, 59200, Israel

Keywords: Secure e-commerce, secure payments, attested delivery, e-banking, cryptographic protocol, non-repudiation.

Abstract: We present a layered architecture for secure e-commerce applications and protocols with fully automated dispute-resolution process, robust to communication failures and malicious faults. Our design is modular, with precise yet general-purpose interfaces and functionalities, and allows usage as an underlying secure service to different e-commerce, e-banking and other distributed systems. The interfaces support diverse, flexible and extensible payment scenarios and instruments, including direct buyer-seller payments as well as (the more common) indirect payments via payment service providers (e.g. banks). Our design is practical, efficient, and ensures reliability and security under realistic failure and delay conditions.

1 INTRODUCTION

Efficient payments are crucial for efficient markets and commerce. Historically, improved payment instruments were critical to the development of commerce and economy. In particular, different secure, unforgeable, authenticated payment tokens, made trading easier: coins are easier to use than barter, paper bills and checks easier than coins, credit-cards often easier than cash and checks.

To facilitate commerce between non-trusting peers payment instruments used trusted third-parties. Often, these parties are trusted to hold the "real" values (*funds*), and transfer them to the payee upon presentment of the appropriate payment authorization or token; this is the traditional role of a bank, or more generally a payment system provider (*PSP*). A separate role for trusted third parties is to prevent or resolve disputes between the parties, including disputes involving bank, PSP, or other payment-related services. Indeed, the ultimate control of payment instruments, prevention of fraud and resolution of disputes, are of the basic attributes of sovereignty. Banks (and some other PSP) are usually subject to laws and government regulation and supervision. One goal of these mechanisms is to prevent, or at least resolve, bank/PSP fraud, e.g. a bank denying a deposit or removing funds without authorization.

Traditionally, the main mechanism for dispute resolution is the use of (handwritten) signatures. Banks provide customers with signed receipt for each deposit, and customers sign authorization for each payment (by check, credit card, or any other means). There are laws and regulations on the necessary authentication and record-keeping by financial institutions, and the system works well, at least for traditional, face-to-face transactions.

Currently, systems deployed with a PSP require complete trust in the PSP, i.e. do not include mechanisms to prove PSP fraud; do not offer receipts or dispute resolution. Moreover, unfortunately and alarmingly, current deployed systems mostly rely on credit card numbers, passwords and other weak-authentication mechanisms, usually transmitted over a connection protected by the SSL or TLS protocol. There were several proposals for deploying additional cryptographic authentication, most notably on (anonymous) digital cash e.g. (Chaum, 1983) and micropayments e.g. (Herzberg, 2003; R. Rivest, 1996). The most significant effort was the *iKP* secure credit-card payments protocol (Bellare et al., 2000), which was adopted (with changes) into the *SET* (Secure Electronic Transactions) system by Visa, MasterCard and others, but not widely deployed. We should note that both *iKP* and *SET* did not include a dispute resolution process.

Several of these proposals, including iKP and SET, authorized transactions using digital signatures by

Herzberg A. and Yoffe I. (2006).
LAYERED ARCHITECTURE FOR SECURE E-COMMERCE APPLICATIONS.
In *Proceedings of the International Conference on Security and Cryptography*, pages 118-125
DOI: 10.5220/0002099801180125

buyers and sometimes also sellers. A possible advantage of signatures over many other authentication mechanisms, is that signatures can be validated by any party - not just the recipient - and at any time, not only at real payment time. In this sense, a payment system can use digital signatures, to allow third-party resolution of disputes, similarly to use of paper signatures. This was definitely one of the goals of signing in iKP.

Currently deployed systems do not have a secure mechanism for fair dispute resolution. To protect consumers, laws and regulations often force a default resolution in favor of the buyer. There are exceptions, most notably, laws and regulations that accept the validity of records kept by a bank, when a client denies having made a transaction. Clearly, such simple dispute-resolution mechanisms provide an acceptable solution to many existing payment scenarios, in spite of their obvious potential unfairness to one party.

In this paper, for the first time, we present a construction which ensures *efficient, fault-tolerant and fair dispute resolution* to e-payment transactions. As mentioned, many of the proposed payment protocols use digital signatures, often with the hope of facilitating dispute resolution. However, few works (Herreweghen, 2000; J. Tang, 2004) including works on non-repudiation (Zhou, 2001; Nenadic and Zhang, 2003) analyzed how the dispute resolution would work in such protocols, and tried to define appropriate process. The resulting processes were designed for a human, and not fully automated. Furthermore, these works did not consider the effects of communication failures.

We considered both *direct* payments, between two parties, and *indirect* payments, involving a trusted third party such as a bank. Furthermore, we divide the payments used in traditional and online commerce, to three groups. The payments could be *non-final, final* and *conditional final*. With non-final payments, the payee need to validate the availability of the payer funds, and take into account the payment might be reversed. For example, paying a merchant with a personal check, may result in the merchant withholding goods, until the check is cleared with the bank. Another example of non-final payments, are credit card MOTO (mail order, telephone order) payments, which are unique in the absence of the credit card itself in the transaction. Most laws allow non-final MOTO payments to be easily reversed, by the customer stating he never approved the order.

On the other hand, with final payments, the approval and availability of payer funds is a function of the payee local validation and actions only. A certified check, for example, from a known bank assures the merchant that he would receive payment if he deposits it within a time frame as defined by banking laws. Another category of payments, are conditional final payments. One would receive payment against a horse betting ticket, which is a certified conditional payment option, only if the horse won the race, or in case of e-mail, if one is signed on a message, it is, conditionally, a payment, if the message is considered as a SPAM (Herzberg, 2004). In Table 1 we summarize the various payment instruments we had mentioned.

Table 1: Payment instruments between two or more parties.

Payment	**Type**	**Example**
Direct (2P)	Immediate	cash, barter, tokens
	Conditional Expirable	betting, futures
Indirect(>2P)	Non-Final (clearing)	MOTO, personal checks, online credit cards usage
	Final	credit cards, certified payment options and checks
	Conditional Final	lottery, escrow, conditional certified payment options

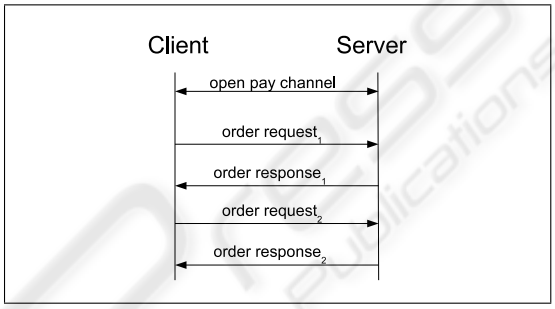

Figure 1: Repetitive successful flow of orders.

The design we present is flexible, and supports several types of payments, allowing its use as underlying layer for many secure commerce protocols. Each payment network principal employs four secure e-commerce application layers, an *application* layer, a *payment* layer, an *order* layer and an *attestation* layer, as a bottom layer, as shown in Figure 2.

The most basic form of payment is a direct, immediate payment order. Our design easily supports such direct payments. See a simple example scenario in Figure 1, where a payments client (*client*) opens a payment channel (*pay channel*) to a payment server (*server*), and then sends several payment requests, to each of which the server sends back a response.

The design also supports conditional and expirable payment orders. In this case, the server responds to the order with a payment *option*. Furthermore, we use the option mechanisms also to facilitate three-party payments, both final and conditional.

One aspect of the flexibility of our layered construction is the support for arbitrary *condition function* on payments, provided as a 'black box' function to payment and e-commerce protocols. The condi-

tion function is defined as part of a *payment agreement* between the payment client and payment server. In particular, we use the same basic two party payment, to enable also payments via a trusted third party PSP, which has a long-term relationship with both buyer and seller. The three party payment transaction (buyer, PSP and merchant) is done by two instances of the simple, two-party transactions, one between buyer and the PSP, and the other between the seller and the PSP. Details and examples are presented later on (specifically see Figure 3). Properly-designed condition functions can support many other scenarios, including betting, hedging and other structured investment devices, and much more.

Related Work. Many payment models and schemes have been developed over the years. Many of these protocols focused on aspects of the payment process, where the widely-used credit card system is not satisfactory. The two main directions here are micropayments (R. Rivest, 1996; Micali and Rivest, 2002; Herzberg, 2003) and digital (anonymous) cash (Chaum, 1983). An important exception is *iKP*, the i-Key-Protocol (Bellare et al., 2000), a family of protocols for secure credit-card payments, which was adopted by MasterCard and Visa for the SET standard (which seems to have been abandoned). Another important exception is the *NetBill* (Cox et al., 1995) protocol, which is a distributed transactional payment protocol featuring atomic delivery, where payment proceeds only if the customer had received the goods. Additional, notable layered architecture, though lacking automated resolution process, is SEMPER (Lacoste et al., 2000), which aimed to create a global, decentralized and secure marketplace. The literature also includes vast research regarding non-repudiation and fair exchange (Nenadic and Zhang, 2003; Pfitzmann et al., 2000; Ray and Ray, 2002) along with dispute resolution (Kremer and Markowitch, 2003) for different levels of a trusted third party involvement (Zhou et al., 1999); for such a survey see (Kremer et al., 2002). The mentioned works lack the proofs to match between orders and issued goods, or don't handle failed submission of orders, payments or payment option deposits, which makes them unsuitable as underlying infrastructure for secure e-commerce services.

Contribution of this work. Our main contribution may be in the presentment of new e-commerce layers as a well defined, fully-automated services, underlying secure e-commerce protocols and applications. We present an architecture, define e-commerce layers with well defined services and interfaces. Another key contribution is our validation constructions, where every e-commerce layer defines its validation functions for automated dispute resolution, which is efficient and fair to all parties.

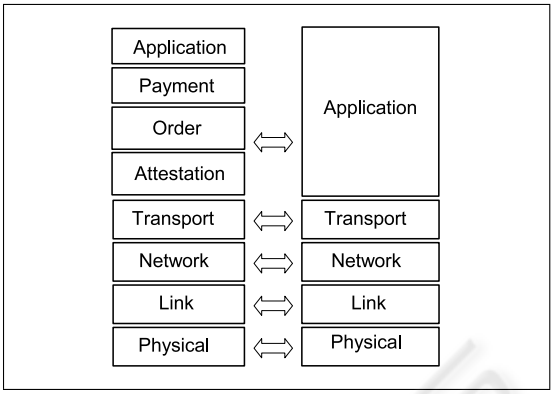

Figure 2: Secure E-commerce layers vs. Internet layers.

Table 2: Attestation evidence structure.

Evidence Field	Description
type	Evidences of origin, delivery and failed submission, *EOO, EOD, EOFS*, respectively.
agr	Attestation agreement.
msg	The message sent.
ci	Creation time interval.
σ	Signature over evidence fields.

2 ATTESTATION LAYER

The *Attestation* layer is the lowest secure e-commerce layer. Attestation layer is based on top of a transport layer, such as, for example, TCP/IP, TLS/SSL, working on top of socket or SSL API, respectively, and provides additional certification services. An attested session is always between three parties: client, server, and a notary (trusted third party), which acts as time-stamping and certification provider.

An *attestation* (Table 2) is a time-stamped and signed statement, by an entity or a notary, on behalf of the entity. An *evidence of delivery* (EOD), is an attestation of message acceptance by an intended message receiver. An *evidence of origin* (EOO), is an attestation of a message origin. An *evidence of failure and submission* (EOFS) is an attestation that the message was correctly sent, yet wasn't acknowledged by the intended receiver (as with EOD).

Generally, the EOD is a proof for the message sender, that the intended message receiver had indeed received the message. The EOO, intended for message receiver, ensures that the message in question had indeed originated from the claimed sender. The EOFS allows the sender to prove he had sent the message in question, even if the message wasn't received due to communication faults or adversarial behavior.

Table 3: Attestation agreement.

Agreement Field	Description
Δ_{bound}	Bound on attestation layer answer, for message delivery.
Id_0, Id_1, *Notary*	The identities of the principals participating in the agreement. Principal's identity is an $(addr, vk)$ tuple, of principal's address and public validation key, respectively.

Time-stamping. The attestation layer supplies, as part of the evidences mentioned above, evidence that a message had existed at specific time interval, and by that time was also signed by the originating party's validation key. This allows cognoscibility of a message even after message originator's validation key had expired, been compromised or revoked.

Confidentiality. In our model we do not treat confidentiality issues, which could be solved by using transport communication layers below attestation layer, for instance, TLS/SSL or IP-Sec layers.

Failed Delivery. We assume simple management of non-delivered messages, where each message is assigned only one type of evidence during lifetime. The attestation layer doesn't try to re-deliver a message (reliability service could be provided by layers below attestation, e.g., TCP), if it had been assigned an EOFS.

Attestation Agreement. An attested communication channel is established using an attestation agreement, specified in Table 3, and for which we assume settings are valid for the whole period the channel remains open. Parties signing the agreement specify own identities and the notary that would be used for attestation. The Δ_{bound}, agreement field, is the bound required for the attestation service to return an evidence, EOD or EOFS, for a sent message.

Validation. The validation functionality is not related to any particular instance of attestation module, and could be invoked by any third party, which had obtained the attested communication agreement and would have to supply the evidence in question. The *Validate(e,type,id)* efficient predicate returns whether the evidence *e*, Table 2, is valid. The *id* specifies the principal whose validation key is to be used from the attestation agreement extracted from the evidence, and *type* specifies the validated evidence type.

Attestation interface. The interface between payment and attestation layers is described in Table 4, and consists of initialization interface and an interface to send and receive message along with their respective evidences.

Initialization. Attestation initialization is two phased, where in *Init*, the attestation layer generates secret and validation key pair, keeps the secret key and in *InitResult* returns the validation key, along with the (communication) address of the attested instance. These values, address and validation key, compose principal's identity, for the above layer, e.g., payment or application layers, and would be used to sign evidences, as specified in Table 2.

3 ORDER LAYER

The layer encapsulates operations ("orders") related to goods and services. The layer provides the service for placing an order for goods or services by a client, and validating that the server returned order result adhere to the order agreement signed between the principals.

Order Agreement. We define an order agreement between trading parties as specified in Table 5. To sign an order agreement, the parties should agree on an appropriate attestation agreement, and a trade validation function, *ValidateTrade(order,goods)*. The function should have *BadOrder, BadGoods, OrderOk, GoodsOk* return values. The *BadOrder* return value is issued for an order which is invalid under the agreement, regardless of the value of goods. The second, *BadGoods* return value, is issued for goods which do not match the order (which should also include the payed amount), the *OrderOk* returned for valid order, without goods; and the *GoodsOk* status is returned when the corresponding goods match the order, in the context of the agreement.

Table 5: Order agreement.

Agreement Field	Description
ValidateTrade(order,goods) returns *status*;	Trade validation function. Validates that issued goods match the order.
AttAgr	Attestation agreement, see Table 3.

Order interface. The interface between the application and order layer, Table 6, defines the initialization, ordering goods or services, and validation of order results. In the first, *Init* phase, each order layer machine establishes its own identity, as returned by attestation layer. Using this information a principal may establish order agreements with other network principals.

When an order channel is established, order transaction are invoked with *Order* event, supplying client specified order information which defines the deposited option or type of goods to acquire, the payment amount, and possibly other relevant information (e.g., original merchant offer). We then expect an *OrderResult* event within finite time, as governed by

Table 4: Attestation layer interface.

Method	Direction	Description
Init(1^k)	in	Initializes attestation layer, with a security parameter.
InitResult(vk,addr)	out	Returns generated validation key *vk* of the initializer, and the principal's address *addr* in the payment network.
OpenChannel(AttestationAgreement)	in	Establishes an attested channel between the source principal, destination principal and a notary.
OpenChannelResult(success)	out	Notifies the principal that an attested channel had been established.
CloseChannel()	in	Closes an attested communication channel.
Send(m)	in	Sends a a message *m* over established, attested, channel.
SendResult(e)	out	Returns an attestation evidence, Table 2, for the sent message.
Deliver(e)	out	Delivers an evidence of origin, Table 2, which also includes the message.

Table 6: Order layer interface.

Method	Direction	Description
Init(k)	in	Initializes the order layer, with security parameter k.
InitResult(vk,addr)	out	Returns initializer's address *addr*, and validation key *vk*.
OpenChannel(OrderAgr)	in	Opens an order channel with the principal(s) specified by *OrderAgr* agreement.
OpenChannelResult(status)	out	Notifies the application of the order channel establishment success.
CloseChannel()	in	Closes an order channel.
Client		
Order(order)	in	Instructs the order layer to issue an order, described by *order*, over an established order channel.
OrderResult(status,result)	out	Returns the order status and result.
Server		
VendRequest(order)	out	Instructs the application layer to issue goods, described by *order*, and implicitly by the order agreement, in the order context.
VendRequestResult(orderresult)	in	Returns the vended goods, from the application.
Validation		
Validate(OrderAgr,orderevidence) returns *result*	in	Validation of order evidences according to the supplied *OrderAgr*. Where *result* is *NoFraud, ServerFraud, ClientFraud*.

the Δ_{bound}, specified in the encapsulated attestation agreement.

On the server side, we assume an application (or upper) level functionality to issue goods or services, using *VendRequest* interface. The goods and services are issued in the context of the order agreement specified for the open order channel, and are verifiable by order agreement's *ValidateTrade*.

The validation functionality, *Validate(e)*, is common to all parties. That is, an automatic dispute resolution system, or an arbiter, upon dispute, would instantiate the order layer, and supply the relevant order agreement along with the the protocol-specific order evidence *e*. Such order layer evidences include evidence of purchase and evidence of failed order, for the client, and evidence of sell, or of a bad client order, for the server. The aforementioned evidences typically composed of pairs of relevant attestation evidences.

4 PAYMENT LAYER

The payment layer encapsulates operations related to funds, which includes payment transactions, final and conditional payment orders, and maintaining of accounts for trading principals.

The payment layer interface is identical to order layer interface, Table 6, with an additional meaning to the *Order* interface method, for support of the aforementioned payment instruments, e.g., ordering a payment option.

Payment layer agreement is specified in Table 7, and includes trading principals initial mutual credit, an encapsulated order agreement and a *ValidateCondition* method for validating payment order deposits.

ValidateCondition. The *ValidateCondition(option,condition)*, is an agreement between parties, regarding evaluation of conditions of certified options at deposit time. It compares option induced condition, with the supplied one, at time of deposit, to decide whether an option should be honored. Using a horse-betting example, an *option*, for instance, could

Table 7: Pay agreement.

Agreement Field	Description
ValidateCondition(option, condition) returns *boolean*;	Validation function for payment options' conditions.
clientCredit	Initial client's credit.
serverCredit	Initial server's credit.
OrderAgr	Order agreement, see Table 5.

specify odds, bet funds, race, the chosen horse, and bookie's signature required to approve the result. The *condition*, supplied along with the option when it is deposited, would be winner's name and race, signed by the bookie specified by the option.

Payment evidences and Validation. Payment layer evidences include balance statements, alongside with evidences of client or third-party's deposits of payment orders, issued to the pay client. We remark that for simplicity of management the statements could be incremental, and enclosed within each payment transactions. Validation of payment layer evidences is straightforward, provided undisputed (signed) server balance statement, for the previous payment transaction, and comparing amount and validity of current transaction evidences which includes server supplied evidences of payment options deposits, to the balance statement of the current (disputed) transaction.

Multi-party payments. Payments which involve multiple PSPs could be facilitated by additional, multi-party, e-commerce layer. The layer would maintain an open, decentralized payment network, which is to provide interoperability among many PSPs without requiring global trust in each PSP, by means of payment routing tables (PRTs) and agreements, that would allow automatic dispute resolution, involving multiple PSPs along a payment path. For further details see (Herzberg, 2003).

5 BREAKDOWN AND USE-CASES

We present few concise use-cases, which together with the presented attestation and payment layer APIs, would clarify how one should implement payment-layer machines (client and server) and common validation functionality, for a payment protocol. We plan to publish full implementation of our order and pay layer protocol, including *Validate* predicates in a separate paper.

1. Ordering goods or services. We begin with a trivial flow of a set of successful orders, as shown in Figure 1. Two trading principals, client and server, use a signed pay agreement to establish a payment channel. Over the payment channel they progress with a set of *request* and *response* pairs, where each such pair, is a successfully completed order. The order request may be requests for goods or services, such as, request for a payment option, and the response contains either a refuse, or the requested goods. The returning goods and services could be validated to match the order and the payed amount, by the client, or any party, e.g., an arbiter, using the pay agreement between the server and the client.

2. Ordering and depositing an option. An option acquisition and deposit, is between two trading parties, namely, server and client. The process begins with ordering an option, where the server removes funds from client's account to pending, and issues the option. The option response arrives to the client, in a form of EOO. When the client deposits such option, as EOO, server validates that it's an EOO issued by itself, validates the option condition, and if the condition is upheld, and option didn't expire, commits the funds transfer from pending to self. Then, a receipt and the goods specified by the option, are sent, in a response message, to the client.

3. Notarized failure of ordering goods or services. We show failed flow of orders, and describe the recovery within the next order, in Figure 5. We begin with the simpler case; in case A, shown in the figure, an order response has failed, with notarization, for server. When the next order arrives, the server includes with the response, the EOFS, for the previous (failed) response, and includes an updated balance statement. The client, which was aware of only one successfully processed order, receives the response and the enclosed EOFS, for the failed response, validates the EOFS, and concludes that both orders were processed, and update its account and balances accordingly.

The next case, described as figure's case B, as opposed to case A, a client's request fails, possibly, because server unavailability, and thus client is issued an EOFS. This EOFS_2 is enclosed with the next, third order, order_3 request. The server, would process all previous failed orders, enclosed with the request. For instance, if the failed order was a deposit of an option, it needs still to be deposited, and the deposit time to be taken from the EOFS. As for failed orders, pay agreement may specify special fees or fines for server's unavailability.

Also, notice the incremental nature of EOFSes, where, for each failing request or response, the next, request or response, respectively, will include all the previous EOFS-es.

4. Communication failures. For benefit of honest parties we supply means of recovery from non-notarized communication failures. Consider the case in Figure 4; a client (or similarly a server) issuing an order request, receives in return a communication failure (instead of an EOD). The client could not possibly know whether the channel had failed, before the re-

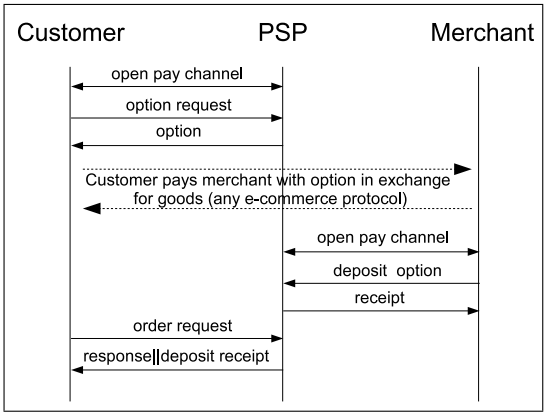

Figure 3: Trade and payments with PSP intermediate.

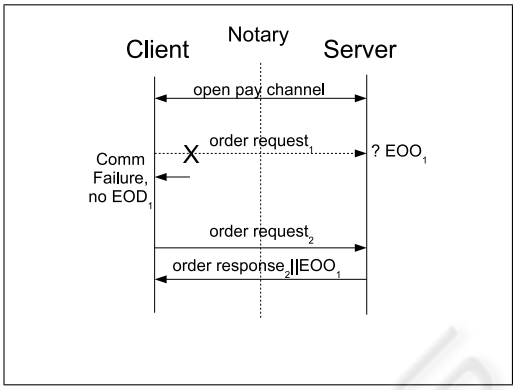

Figure 4: Non-notarized communication failure for orders.

quest had been delivered (and server had obtained an EOO), or afterward, and the failure had prevented him from receiving an EOD (or EOFS). Thus, the state of the order would be unknown, until the next order successfully completes. As could be seen in the figure, if the order was indeed received by a honest server, the client would see the order's EOO included in the next response, and update order's status, and account balances, accordingly.

5. Dispute resolution. Consider disputes regarding amount of funds in the accounts, after a series of orders. Since account balances could be validated incrementally, adhering to the concept of retaining last evidences, the client would turn to an arbiter with two subsequent sever response evidences, as a payment evidence. The arbiter would instantiate the payment, order and attestation layers, given the pay agreement, and would invoke the corresponding *Validate* routine. Possible implementation of the routine, which is shared by all protocol parties, would be taking the balance statement from the (last) non-disputed server response, and comparing all the reported order-results and deposit balance changes, to the balance reported in the current response, in addition to checking deposits validity with *ValidateCondition*.

Additional scenarios, e.g., server just ignoring an order, for which the client had obtained an EOD, could be addressed by introducing additional order layer evidences and order layer trusted party, which could, similarly to attestation, issue an evidence of failed (or ignored) order for the client.

6. Payments with PSP intermediate. With indirect final payments, the payee (merchant) receives a pre-authorized payment option from the payer (customer). The payee is able to locally validate the payment, unlike payments which require clearance to ensure availability of payer funds. When the option is deposited to the PSP, its condition is evaluated, and if the evaluation succeeds the PSP makes the transfer

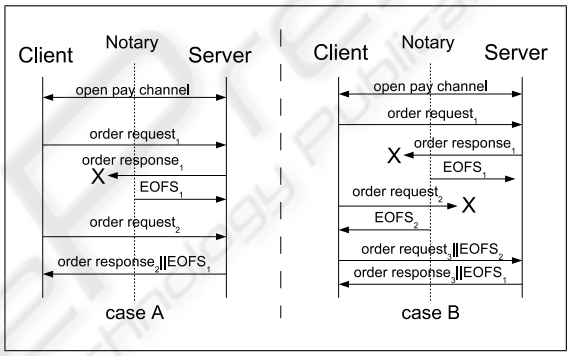

Figure 5: Failed order flows.

between payer an payee accounts, otherwise the payment option is discarded. In Figure 3 we show our schematic settings for such indirect payments. The conditional payments flow between the payer and the payee progress as two separate interactions, as follows: a PSP receives an order for an option from the payer, moves the funds specified by the order to pending, and returns a payment option. The payer, in a process of trade with the payee, handles the payment option to the payee. In his own pay interaction against the PSP, the payee deposits the option. At deposit, the PSP checks for expiration and option condition, and then two receipts are generated, one indicating the end of PSP's transaction against the payer and the other indicating the end of PSP's transaction against the payee.

The trade itself between the payer and the payee, shown as a dotted line, could be performed by any e-commerce protocol. Although the merchant and the customer do not maintain long-term relationship as required in our settings (such long-term relationship renders introduction of a PSP intermediate as artificial), the option itself could be used as a material

for signing ad-hoc pay agreement (see Section 4) between the parties. Then, multiple trade transactions could occur between payer and payee, until the credit specified by the option is exhausted.

In addition, using conditional certified payment options provides extra flexibility. Consider, for example, the case of assuring goods delivery, as with NetBill (Cox et al., 1995). The PSP conditions the payee presenting a delivery receipt signed by a customer, or notarized delivery services (N. Asokan, 2000). Such receipt would be checked by the *ValidateCondition* pay agreement predicate, upon option deposit, assuring option deposit is successful only when goods had been delivered.

6 CONCLUSION

Following our construction we are implementing a payment protocol with automated dispute resolution, it would be published in a separate paper. We also consider applying our constructions and protocol for the SICS (Herzberg, 2004), SPAM-prevention system. We believe SICS could be easily adapted for the presented architecture, in collecting and resolving SPAM-related payments.

In conclusion, we have introduced a novel, agreement-based, mechanism, and have shown how to use it to construct final and conditional final payments, between two parties which maintain a long term relationship, or how to conduct trade when a PSP is a trusted party. Our constructions are practical, layered, with promise of automatic dispute resolution based on precise agreements and relatively simple cryptographic constructions assumed.

REFERENCES

Bellare, M., Garay, J., Hauser, R., Herzberg, A., Krawczyk, H., Steiner, M., Herrenweghen, E. V., and Waidner, M. (2000). Design, Implementation and Deployment of the iKP Secure Electronic Payment System. In *Journal on Selected Areas in Communication, special issue on Network Security*, volume 18, pages 611–627.

Chaum, D. (1983). Blind Signatures for Untraceable Payments. In *Advances in Cryptology - Proceedings of CRYPTO '82*, pages 199–203. D. Chaum, R. L. Rivest, and A. T. Sherman, Eds., Plenum, NY.

Cox, B., Tygar, J. D., and Sirbu, M. (1995). NetBill security and Transaction Protocol. In *The First USENIX Workshop on Electronic Commerce*, pages 77–88.

Herreweghen, E. V. (2000). Non-repudiation in SET: Open Issues. In *Proceedings of the 4th Conference on Financial Cryptography*.

Herzberg, A. (2003). Payment technologies for E-commerce, *Chapter 13, Micropayments*. Springer-Verlag.

Herzberg, A. (2004). Controlling Spam by Secure Internet Content Selection. In *Proceedings of Secure Communication Networks (SCN)*, volume 3352 of *LNCS*, pages 337–350. Springer-Verlag.

J. Tang, A. Fu, J. V. (2004). Supporting Dispute Handling in E-commerce Transactions, a Framework and Related Methodologies. In *Electronic Commerce Research Journal*, volume 4, pages 393–413. Kluwer Academic.

Kremer, S. and Markowitch, O. (2003). Fair Multi-Party Non-Repudiation Protocols. *International Journal on Information Security*, 1(4):223–235.

Kremer, S., Markowitch, O., and Zhou, J. (2002). An Intensive Survey of Non-repudiation Protocols. *Computer Communications*, 25(17):1606–1621.

Lacoste, G., Pfitzmann, B., Steiner, M., and Waidner, M., editors (2000). *SEMPER - Secure Electronic Marketplace for Europe*, volume 1854 of *Lecture Notes in Computer Science*. Springer-Verlag.

Micali, S. and Rivest, R. (2002). Micropayments revisited. In *Progress in Cryptology — CT-RSA 2002*, volume 2271 of *LNCS*. In Bart Preneel, editor, Springer-Verlag.

N. Asokan, V. Shoup, M. W. (2000). Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, 18:593–610.

Nenadic, A. and Zhang, N. (2003). Non-repudiation and Fairness in Electronic Data Exchange. In *Proceedings of 5th International Conference on Enterprise Information Systems (ICEIS)*, pages 55–62, Angers, France.

Pfitzmann, B., Schunter, M., and Waidner, M. (2000). Provably Secure Certified Mail. In *IBM Research Report RZ 3207 (#93253), IBM Research Division*, Zurich.

R. Rivest, A. S. (1996). PayWord and MicroMint: Two Simple Micropayment Schemes. In *Proceedings of the International Workshop on Security Protocols*, pages 69–87.

Ray, I. and Ray, I. (2002). Fair exchange in E-commerce. *SIGecom Exch.*, 3(2):9–17.

Zhou, J. (2001). *Non-repudiation in electronic commerce*. Computer Security Series. Artech House.

Zhou, J., Deng, R. H., and Bao, F. (1999). Evolution of Fair Non-repudiation with TTP. In *ACISP '99: Proceedings of the 4th Australasian Conference on Information Security and Privacy*, pages 258–269, London, UK. Springer-Verlag.