RZ 3156 (# 93202) 06/30/99
Computer Science/Mathematics 17 pages

Research Report

Using Digital Signatures as Evidence of Authorizations in Electronic Credit-Card Payments

Els Van Herreweghen

IBM Research
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

Research
Almaden · Austin · Beijing · Haifa · T.J. Watson · Tokyo · Zurich

Using Digital Signatures as Evidence of Authorizations in Electronic Credit-Card Payments

Els Van Herreweghen

IBM Research, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland

Abstract

Credit-card payment protocols such as *i*KP and SET use digital signatures to authenticate messages and authorize transactions. It is assumed that these digital signatures make the parties' authorizations non-repudiable, i.e., provable to a third-party verifier. In this paper, we define a set of statements which participants in a payment may want to prove. We then investigate which of these statements can be proved with the digital signatures within *i*KP and SET, and make some general comments about requirements for provability in payment protocols.

1 Introduction

Digital signatures in the *i*KP [BGH+95] payment protocol are intended, by design, to represent non-repudiable message receipts for customer, merchant and bank. The SET [MV97] protocols are similar in design to *i*KP; though the SET specifications do not claim non-repudiability of the SET signatures, press releases and public opinion tend to attach this feature to them. In this paper, we investigate the value of *i*KP and SET digital signatures as evidence towards an external verifier. Such a verifier may be a subsystem of a third-party dispute handler arbitrating payment disputes, or an online ombuds service as in [Kat96]. Alternatively, it can be a standalone system used by a company's accounting department or a national tax department to verify digital receipts submitted as evidence of transactions. The latter use illustrates the need for digital receipts to be self-contained; they should be usable as valid transaction receipts, without the need for additional evidence collected from semi-trusted parties such as banks.

A fair amount of work has been done recently on formal specification and verification of e-commerce protocols [Kai95, KN98, Bol97, MS98, Bra97]. Both [Kai95] and [KN98] focus on accountability, by introducing authentication logics modified with predicates of the form "A can prove X to B". The statements X in both logics are payment system dependent; [KN98] apply their logic to SET and prove the following: a Merchant M, having received a PReq (Payment Request) message from Customer C with Order Instructions OI, can derive that *M believes M canprove (C said OI) to J*, where J is an independent verifier.

In this paper, we define a number of concrete, protocol-independent assertions that parties participating in a payment protocol may want to prove; we then evaluate the evidence collected by the two example protocols as to their usability in proving those statements. We do not prove the protocols to be secure, in the sense of resistance against outsider and replay attacks. Rather, we focus on provability of the respective authorizations to an external verifier.

In Section 2, we describe a high-level protocol representing both *i*KP and SET. In Section 3, we describe what kind of statements each participant in this payment protocol may want to prove, using the evidence collected during a payment run. These statements form a subset of the generic payment claim language described in [AHS98]. In Sections 4 and 5, we then describe in more detail the respective protocols *i*KP and SET, and evaluate their evidence tokens against the requirements described in Section 3. Section 6 summarizes our findings and concludes that secure authentication using digital signatures does not guarantee provability.

2 High-level payment protocol description

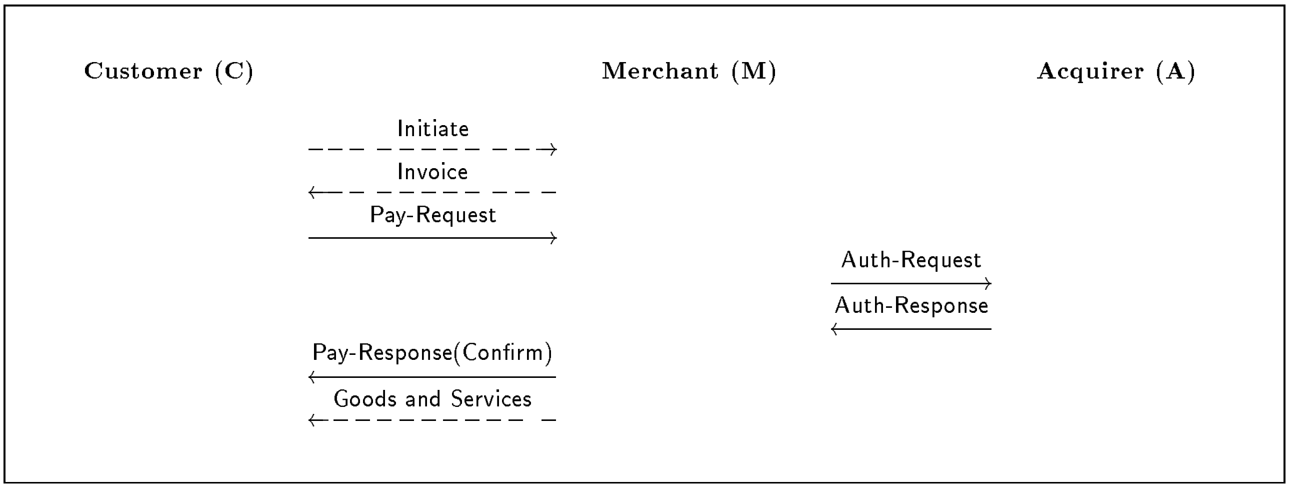

Figure 1: Generic credit-card payment protocol

The *i*KP and SET protocols both can be represented by the high-level protocol depicted in Figure 1. Before the start of the actual payment protocol, Customer and Merchant have agreed on a description and price of the goods to be purchased. In an optional Initiate - Invoice exchange, Customer and Merchant exchange variables, options and randomizers needed in the ensuing payment messages. The Customer then sends a Pay-Request message to the Merchant which the Merchant then uses to produce an Auth-Request asking authorization from the Acquirer. The Acquirer goes through the financial network to obtain the authorization and returns an Auth-Response to the Merchant, indicating success or failure, and optionally the actually captured amount. The Merchant then produces a Pay-Response (Confirm) message and sends it to the Customer.

Both *i*KP and SET actually represent a family of different protocols depending on which parties (all, only Merchant and Acquirer, only Acquirer) digitally sign messages. This paper aims at analyzing the most secure versions in both protocol families; therefore, we will in both cases analyze the protocol variant where all parties digitally sign messages.

A protocol option set by Merchant (eventually also Customer) determines whether the Auth-Request - Auth-Response exchange results in the actual transfer of money (simultaneous authorization and capture) or whether an ensuing Cap-Request - Cap-Response exchange between Merchant and Acquirer completes the actual transaction (separate authorization and capture). To simplify the analysis, we will assume the former scenario, i.e. simultaneous authorization and capture. For the same reason, we will not consider reversal or refund of payments.

3 Requirements for provability

In [AHS98], a language was presented to express claims (or provable statements) that participants in a payment protocol may want to be able to prove. This language is derived from the generic payment service definition in [APASW98] and is independent of the payment model (stored-value, account-based). A payment transaction involves payer (Customer, C), payee (Merchant, M) and bank (Acquirer, A) and defines three types of *primitive transactions*, each representing the view of a subset of players on a payment transaction:

- In *value subtraction*, a user allows the bank to remove "real value" from the user; this implies the user's right to spend "electronic value."
- In *value claim*, a user requests that the bank gives "real value" to the user.
- In *payment*, the payer transfers value to the payee.

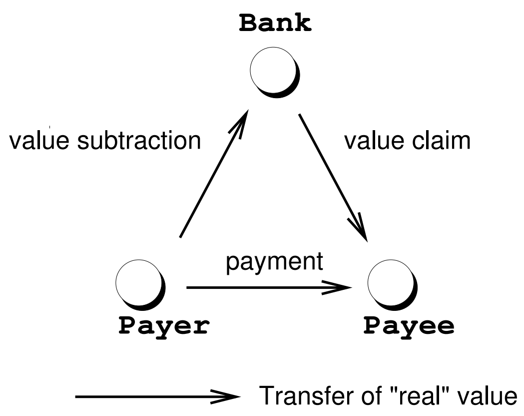

Figure 2: Value Transfer Transactions

In stored-value (electronic purse) systems, *value subtraction*, *payment* and *value claim* may be separate transactions, taking place at different points in time. In the account-based protocol in Figure 1, where the Acquirer is involved online and authorization and capture occur simultaneously, *value subtraction*, *payment* and *value claim* complete at the same time and cannot be separated. Distinguishing between the three *primitive transactions*, however, enables us to formulate disputes between each subset of players: it allows us to reason about the bank's behavior as well, as it receives and sends authorization messages to the other parties.

In the remainder of this section, we describe different statements participants in a payment may want to prove. In Section 3.1, we describe which authorizations ("Party **authorized** *primitive transaction*") parties may want to prove. These are the fundamental provability requirements since they define which value we would like to attach to each of the digital signatures in the payment protocol. Section 3.2 looks at additional statements parties may want to prove in the case of unsuccessful or interrupted transactions.

3.1 Proving authorizations of *primitive transactions*

After completion of the payment protocol in Figure 1, each of the parties (Customer C, Merchant M and Acquirer A) ideally has non-repudiable proof of transaction authorization by the two other parties. Using a notation similar to the one used in [AHS98], following requirements should then be fulfilled:

- M can prove "C **authorized** *payment* (C, M, Amount, Date, Ref)".
  Ref stands for a set of reference parameters which may be protocol specific, such as transaction identifiers or description of the goods purchased. We will further abbreviate the statement "C **authorized** *payment* (...)" to "Auth(C-M)".
- C can prove "M **authorized** *payment* (C, M, Amount, Date, Ref)", or short: "Auth(M-C)".
- A can prove "C **authorized** *value subtraction* (C, A, Amount, Date, Ref)", short: "Auth(C-A)".
- C can prove "A **authorized** *value subtraction* (C, A, Amount, Date, Ref)", short: "Auth(A-C)".
- A can prove "M **authorized** *value claim* (M, A, Amount, Date, Ref)", short: "Auth(M-A)".
- M can prove "A **authorized** *value claim* (M, A, Amount, Date, Ref)", short: Auth(A-M)".

For a protocol following the scheme in Figure 1, Pay-Request must then contain proof of Auth(C-M) and Auth(C-A); Auth-Request contains proof of Auth(M-A) and forwards proof of Auth(C-A); Auth-Response contains proof of Auth(A-M) and Auth(A-C); Pay-Response, finally, contains proof of Auth(M-C) and forwards proof of Auth(A-C).

The statements that can be proved by the different parties thus evolve in time as follows:

- After receiving Pay-Request, M can prove Auth(C-M);
- After receiving Auth-Request, A can prove Auth(C-A) and Auth(M-A);
- After receiving Auth-Response, M can prove Auth(A-M) (in addition to Auth(C-M));
- After receiving Pay-Response, C can prove Auth(A-C) and Auth(M-C).

If all of the above requirements are fulfilled, all the parties in a completed transaction have sufficient proof to convince a verifier of their view of the primitive transactions in which they were involved.

The absence of one of these provable authorizations need not make the payment protocol insecure; but it may make disputes between the parties involved in the specific *primitive transaction* more difficult (e.g., having to rely on evidence provided by other parties) or even impossible without resorting to off-line means (e.g., an account statement proving a debit and thereby a payment). In the following, we highlight the importance of the respective authorizations:

- Auth(M-C) is equivalent to a receipt of payment by M. In its absence, C cannot prove the payment to M without involving evidence collected by A, or without resorting to off-line means (e.g., account statement).
- Auth(A-C) is equivalent to a receipt by A of the amount deducted from C's account and can eventually be replaced by an off-line account statement. It allows C to verify that the amount deducted by A and the amount in M's receipt are equal.
- Auth(C-M) is obviously important if M delays the actual authorization/capture and makes certain decisions (such as shipping goods, or reserving stock) based on Auth(C-M) before contacting A. This authorization may seem less important in the case where M directly captures the money and therefore does not need Auth(C-M) as a payment guarantee. However, Auth(C-M) still has its value as a proof of terms and conditions of the payment, such as description of the goods (which, in case of disagreement, M may otherwize not be able to prove).
- Auth(C-A) is A's proof that C entitled A (or its own issuing bank through A) to deduct money from C's account.

- Auth(M-A) is A's proof that M asked the payment to be made to him.
- Auth(A-M) is M's proof that A transferred (or committed to transfer) the money to M's account.

When analyzing *i*KP and SET (in Sections 4 and 5), we will investigate for each of the above authorizations whether the interested party has the necessary evidence to prove the authorization, whether it needs other parties to cooperate in giving evidence in order to prove the authorization, or whether the statement cannot be proved at all. We will see that *i*KP gives each party enough evidence to prove the other parties' respective authorizations. In SET, on the other hand, some of the authorizations cannot be proved or can only be proved with the help of evidence collected by other parties.

3.2 Fairness Considerations

The design of the *i*KP and SET protocols (represented in the generic payment protocol in Figure 1) defines C to be the initiator of the *payment*, i.e., C authorizes *payment* before M (the *payment* responder) does. C also is the initiator in *value subtraction*, and M is the initiator in *value claim*.

This inherent asymmetry is a possible source of unfairness: if the initiator of a *primitive transaction* does not receive a reply, it may not know whether its message arrived, whether it arrived but will not be replied to or followed up on, or whether the responder followed up on the payment but refuses to issue a response.

Consider the example of C having sent Pay-Request with a certain validity duration, and not receiving any response. Without resorting to off-line mechanisms, such as calling its bank, C has no way to determine whether M captured the payment. Even if the bank confirms the payment, this evidence is not equivalent to a formal receipt by the Merchant, since C cannot consider it as sufficient evidence in a dispute with M: M can always claim the bank made a mistake. Moreover, if C knows the payment was not captured, it does not know whether M will still capture it within the validity time of the Pay-Request. Therefore, C does not know whether it should place the same order with another merchant, or to place a new order with the same M (possibly now to different conditions).

Unfairness due to network failures can be avoided by building fault tolerance mechanisms into the payment system, allowing parties on the initiating side of an exchange (C in Pay-Request, M in Auth-Request) to keep polling for a reply.

Unfairness due to dishonest parties can be partially compensated for in an infrastructure for dispute handling. The following paragraphs describe some different, complementary, mechanisms.

- Forced transaction termination

  In the presence of a dispute handling or arbitration infrastructure, C, when uncertain about the outcome of the transaction, may contact the arbitrator and ask the arbitrator to extort a (positive or negative) receipt from M. This can be a protocol-compliant receipt, or a signed statement from M that M has no record of the transaction. Refusal by M to make a statement results in the arbitrator composing an affidavit stating the payment did not occur. This is a mechanism similar to the one described in [ASW97].

- Replacement receipts

  Another, complementary, mechanism for limiting unfairness is to define which pieces of evidence can be used as a replacement receipt for a specific authorization. In the case of C not receiving Pay-Response, the arbitrator may ask evidence from A. The arbitrator may conclude that a valid Auth-Request-Auth-Response pair provided by A is sufficient evidence to prove M's authorization of the payment, and that these pieces of evidence are equivalent to Auth(M-C).

- Proving receipt of authorizations

  In some cases, the sender of an authorization may simply want to prove that another party received its authorization. This may be the case if the receiver did not execute the protocol as agreed or required:

  - C may want to prove that M received and was able to verify C's *payment* authorization: "M **received** (C **authorized** *payment* (C,M,Amount,Date,Ref))".
  - M may want to prove that A received and was able to verify M's *value claim* authorization: "A **received** (M **authorized** *value claim* (M,A,Amount,Date,Ref))".

- C may want to prove that A received and was able to verify C's *value subtraction* authorization: "A **received** (C **authorized** *value subtraction*))".

Proving these statements may help the claimer in proving higher-level claims in the language defined in [AHS98] about what parties **could** have done. Consider, as an example, the case where C initiated a payment based on a special offer by M, and M didn't reply to this payment. C can try to prove that M **received** (C **authorized** *payment* (C,M,Amount,Date,Ref), and thus M **could** have captured C's payment.

In order to prove receipt of a certain payment message, one may consider involving third-party services, such as certified e-mail or fair exchange services [ASW97]. These can provide the sender of a payment message with a signed receipt (by the receiver) of the message, proving that the recipient received this specific message. It is sometimes assumed that such a signed receipt also suffices to prove that the recipient could continue the protocol. Applied to the above example, one could assume that a signed receipt by M of C's valid Pay-Request message proves that "M **received** (C **authorized** *payment* (...))". When analyzing *i*KP and SET, we will specifically look at this possibility: we will investigate whether a signed receipt of a certain message indeed proves that the recipient had all the necessary data to verify the message (and thus was able to continue a normal protocol run).

Forced termination of a transaction is a mechanism outside of the payment protocol, and will not be discussed with specific reference to *i*KP and SET. As for replacement receipts, an arbitrator can, as a general rule, derive Auth(M-C) from Auth(M-A) and Auth(A-M). The applicability of the rule depends on the provability of the latter statements. Also replacement receipts will not be discussed further.

Using signed message receipts to prove receipt of authorizations poses certain requirements on the payment protocol. We will therefore come back to this issue when analyzing *i*KP and SET in Sections 4 and 5.

4 *i*KP

4.1 *i*KP Protocol Overview

Figure 3 shows the most complete version of the *i*KP protocols as implemented in the Z*i*P [BGH+99] prototype: 3KP with mandatory Confirm message containing the Acquirer's signature Sig_A [1].

The only protocol option which is not yet fixed by the above choice is PFLAGS : $\mathrm{SIG_M}$, i.e., whether or not a signature Sig_M of the Merchant is sent to the Customer in Invoice (representing a signed Invoice) or only in Cancel or Confirm. We will discuss both variants and their influence on provability.

The resulting protocol and the description of its cryptographic primitives and components are depicted in Table 1 and Figure 3.

$\mathcal{H}(\cdot)$	A strong collision-resistant one-way hash function.
$\mathcal{E}_X(\cdot)$	Public-key encryption, providing also message integrity, using X's public key PK_X.
$\mathcal{S}_X(\cdot)$	Signature with respect to X's secret key SK_X. $\mathcal{S}_X(M)$ does not include message M.

Table 1: Cryptographic Primitives in *i*KP

Public-key certificates are transported outside of the *i*KP messages. The Cancel message is used if the Merchant decides not to go ahead with payment authorization and may not be used if the Merchant has sent an Auth-Request. For further details, we refer to [BGH+99].

In the following sections, we now describe which pieces of evidence can prove the statements described in Sections 3.1 and 3.2. We do so by formulating "rules" which a verifier V, when presented with pieces of *i*KP evidence, can use to infer conclusions about those statements. A rule of the form

[1] [BGH+99] uses the terms Buyer and Seller. We changed the *i*KP notation to be consistent with our (Customer, Merchant) notation.

- **Atomic and Composite Fields:**

SALT_D	Random number generated by the Customer, used to salt DESC.
SALT_C	Random number used to salt the account number in the Customer's certificate.
PRICE	Amount and currency
INV-EXP	Invoice (offer) expiration specified by the Merchant.
DATE	Merchant's date/time stamp, used for "coarse grained" replay protection.
NONCE_M	Merchant's nonce (random number) used for more "fine grained" replay protection.
ID_M	Merchant id. This identifies Merchant to Acquirer.
TID_Z	Transaction ID assigned by each party to an *i*KP transaction. Generated at a layer above *i*KP and not explicitly carried in *i*KP messages.
DESC	Description of purchase/goods.
CAN	Customer's Account Number (e.g., credit card no.). Includes expiration date.
ID_C	Customer's randomized pseudo-ID. $\mathrm{ID}_C = \mathcal{H}(\mathrm{R}_C, \mathrm{CAN})$ with R_C a random number chosen by the Customer.
RESPCODE	Authorization response from the Acquirer. Can also be set by the Merchant in the case of a cancellation.
AUTHTIME	Authorization time set by the Acquirer.
OPTSIG_Z	Optional text for inclusion in signatures. Not explicitly carried in *i*KP messages.
V	Random number generated by Merchant for use as a proof that Merchant has accepted payment (i.e., to bind positive Confirm and Invoice messages).
VC	Random number generated by the Merchant, used to bind negative Confirm /Cancel and Invoice messages.
Common	TID_M, TID_C, PFLAGS, PRICE, ID_M, DATE, INV-EXP, NONCE_M, ID_C, $\mathcal{H}(\mathrm{DESC}, \mathrm{SALT}_D)$, $\mathcal{H}(\mathrm{V})$, $\mathcal{H}(\mathrm{VC})$
Clear	PFLAGS, ID_M, DATE, NONCE_M, $\mathcal{H}(\mathrm{Common})$, INV-EXP, $\mathcal{H}(\mathrm{V})$, $\mathcal{H}(\mathrm{VC})$
EncSlip	$\mathcal{E}_A(\mathrm{PRICE}, \mathcal{H}(\mathrm{Common}), \mathrm{CAN}, \mathrm{R}_C, \mathrm{SALT}_C)$
Sig_M	$\mathcal{S}_M(\mathcal{H}(\mathrm{Common}), \mathrm{OPTSIG}_M)$
Sig_C	$\mathcal{S}_C(\mathrm{EncSlip}, \mathcal{H}(\mathrm{Common}), \mathrm{OPTSIG}_C)$
Sig_A	$\mathcal{S}_A(\mathrm{RESPCODE}, \mathrm{AUTHTIME}, \mathcal{H}(\mathrm{Common}), \mathrm{OPTSIG}_A)$

- **Protocol Flows:**

Initiate:	C	$\xrightarrow{\mathrm{PFLAGS},\ \mathrm{SALT}_D,\ \mathrm{ID}_C}$	M
Invoice:	C	$\xleftarrow{\mathrm{Clear},\ [\mathrm{Sig}_M]}$	M
Payment:	C	$\xrightarrow{\mathrm{EncSlip},\ \mathrm{Sig}_C}$	M
Cancel:	C	$\xleftarrow{\mathrm{RESPCODE},\ \mathrm{VC},\ [\mathrm{Sig}_M]}$	M
Auth-Request:	M	$\xrightarrow{\mathrm{Clear},\ \mathcal{H}(\mathrm{DESC}, \mathrm{SALT}_D),\ \mathrm{EncSlip},\ \mathrm{Sig}_M,\ \mathrm{Sig}_C}$	A
Auth-Response:	M	$\xleftarrow{\mathrm{RESPCODE},\ \mathrm{AUTHTIME},\ \mathrm{Sig}_A}$	A
Confirm:	C	$\xleftarrow{\mathrm{RESPCODE}\ (=\mathrm{positive}),\ \mathrm{AUTHTIME},\ \mathrm{V},\ [\mathrm{Sig}_M],\ \mathrm{Sig}_A}$	S
		—OR—	
Confirm:	B	$\xleftarrow{\mathrm{RESPCODE}\ (=\mathrm{negative}),\ \mathrm{AUTHTIME},\ \mathrm{VC},\ [\mathrm{Sig}_M],\ \mathrm{Sig}_A}$	S

Figure 3: 3KP payment with in-line authorization

evidence (Evidence1, Evidence2, ...),
verify (Statement1, Statement2, ...)
$\rightarrow$ **conclusion**

expresses that, if V is given the pieces of evidence in **evidence**, and if V also can verify statements (equalities, cryptographic relationships) in **verify**, then V will arrive to **conclusion** (or, equivalently: *then* **conclusion** can be proved to V).

In the following analysis, we make abstraction of the linking between keys and entities: if message M is signed with a secret key K_s of which the public counterpart K_p is certified by a trusted Certification Authority as belonging to entity E, then M is signed by E.

We also make abstraction of explicit distribution and verification of the public-key certificates certifying and carrying the K_p's; in the following, when the verifier can convince itself that P signed message M at time T, V has been presented with a certificate for P's public key K_p which was valid at time T.

4.2 Proofs of Authorizations

4.2.1 M proving Auth(C-M)

After receiving Payment, the Merchant has all the evidence needed to prove Auth(C-M). Evidence required consists of the contents of Payment, together with HCOMMON (which should equal $\mathcal{H}$(Common)), Common and OPTSIG_C. Note that Common consists of data from the negotiation phase (PRICE), from Initiate(PFLAGS, ID_C) and from Invoice (Clear).

evidence (OPTSIG_C, HCOMMON, Sig_C, EncSlip, Common [TID_M, TID_C, PFLAGS, PRICE, ID_M, DATE, INV-EXP, NONCE_M, ID_C, HDESC, HV, HVC]),
verify (HCOMMON $= \mathcal{H}$(Common),
$\text{Sig}_C = \mathcal{S}_C$(EncSlip, HCOMMON, OPTSIG_C))
$\rightarrow$ C **authorized** *payment* (ID_C, ID_M, PRICE, DATE, Ref(Common [HDESC, ...])).

OPTSIG_C has to be stored by M and provided to V in order to prove C's authorization. Though not transmitted as part of the protocol, OPTSIG_C is thus a mandatory part of the evidence assembled by S. In addition, OPTSIG_C needs to be forwarded to A in order for A to verify and prove Auth(C-A) (Section 4.2.3).

The verifications (**verify**) performed on the evidence convince V that C signed the claimed data (EncSlip, HCOMMON, OPTSIG_C); and, through HCOMMON, C signed the data in Common. V thus concludes that C authorized the payment with the Ref parameter containing the data in Common (specifically HDESC). Ref() is an extendable list of reference data.

The previous rule does not prove the description DESC of the goods for the transaction: it only proves authorization of *payment* with $\mathcal{H}(\text{DESC}, \text{SALT}_D) = \text{HDESC}$. If M wants to prove DESC, it can do so by also providing DESC and SALT_D. This is shown in the following extension rule, which can be used in conjunction with Auth(C-M) and Auth(M-C) authorizations:

evidence (DESC, SALT_D),
verify (X **authorized** *payment* (Y, Z, PRICE, DATE, Ref(Common [HDESC, ...])),
HDESC $= \mathcal{H}(\text{DESC}, \text{SALT}_D)$)
$\rightarrow$ X **authorized** *payment* (Y, Z, PRICE, DATE, Ref(Common, DESC)).

S can thus choose not to disclose (and prove) DESC while still able to prove Auth(C-M) for a certain PRICE. This is an illustration of an *i*KP design goal, namely to disclose a minimal amount of data to achieve a certain proof.

4.2.2 C proving Auth(M-C)

After receiving a positive Confirm, C can prove M's *payment* authorization:

> **evidence** (OPTSIG$_M$, HCOMMON, Sig$_M$, V, Common [TID$_M$, TID$_C$, PFLAGS, PRICE, ID$_M$, DATE, INV-EXP, NONCE$_M$, ID$_C$, HDESC, HV, HVC]),
> **verify** (HCOMMON $= \mathcal{H}$(Common),
> Sig$_M$ $= \mathcal{S}_M$(HCOMMON, OPTSIG$_M$),
> HV $= \mathcal{H}(V)$)
> $\rightarrow$ S **authorized** *payment* (ID$_C$, ID$_M$, PRICE, DATE, Ref(Common)).

Alternatively, after having received a Cancel or negative Confirm, C can use VC to prove that M refused or cancelled payment (or, equivalently, that M authorized a Payment of amount $= 0$):

> **evidence** (OPTSIG$_M$, HCOMMON, Sig$_M$, VC, Common [TID$_M$, TID$_C$, PFLAGS, PRICE, ID$_M$, DATE, INV-EXP, NONCE$_M$, ID$_C$, HDESC, HV, HVC]),
> **verify** (HCOMMON $= \mathcal{H}$(Common),
> Sig$_M$ $= \mathcal{S}_M$(HCOMMON, OPTSIG$_M$),
> HVC $= \mathcal{H}(VC)$)
> $\rightarrow$ S **authorized** *payment* (ID$_C$, ID$_M$, 0, DATE, Ref(Common)).

Both previous rules can be extended to include DESC, using the extension rule in Paragraph 4.2.1.

4.2.3 A proving Auth(C-A)

With Auth-Request, A can prove C's authorization of *value subtraction*. Note that also OPTSIG$_C$, even if not intended for A, needs to be forwarded from M to A, in order for A to verify Sig$_C$ and to verify and prove Auth(C-A).

> **evidence** (OPTSIG$_C$, HCOMMON, Sig$_C$, EncSlip, Common [TID$_M$, TID$_C$, PFLAGS, PRICE, ID$_M$, DATE, INV-EXP, NONCE$_M$, ID$_C$, HDESC, HV, HVC]),
> **verify** (HCOMMON $= \mathcal{H}$(Common),
> Sig$_C$ $= \mathcal{S}_C$(EncSlip, HCOMMON, OPTSIG$_C$))
> $\rightarrow$ C **authorized** *value subtraction* (ID$_C$, ID$_M$, PRICE, DATE, Ref(Common)).

The above rule does not use EncSlip as evidence. In fact, the contents of EncSlip are not meant to be used in a proof to an external verifier, since they are encrypted before being signed by C. The function of EncSlip is to allow A to retrieve necessary values (such as PRICE) to rebuild Common, to verify the correctness of credit card information and the linking between this information and ID$_C$ (using R_C in EncSlip).

4.2.4 C proving Auth(A-C)

With a positive Confirm and OPTSIG$_A$ (forwarded by S), C can prove A's authorization of *value subtraction*:

> **evidence** (OPTSIG$_A$, Sig$_A$, HCOMMON, RESPCODE, AUTHTIME, Common [TID$_M$, TID$_C$, PFLAGS, PRICE, ID$_M$, DATE, INV-EXP, NONCE$_M$, ID$_C$, HDESC, HV, HVC]),
> **verify** (RESPCODE is positive,
> HCOMMON $= \mathcal{H}$(Common),
> Sig$_A$ $= \mathcal{S}_A$(RESPCODE, AUTHTIME, HCOMMON, OPTSIG$_A$))
> $\rightarrow$ A **authorized** *value subtraction* (ID$_C$, A, PRICE, AUTHTIME, Ref(Common)).

A similar rule can be formulated for denied authorizations (negative RESPCODE), in which case A authorizes an amount=0:

evidence (OPTSIG$_A$, Sig$_A$, HCOMMON, RESPCODE, AUTHTIME,
Common [TID$_M$, TID$_C$, PFLAGS, PRICE, ID$_M$, DATE,
INV-EXP, NONCE$_M$, ID$_C$, HDESC, HV, HVC]),
verify (RESPCODE is negative,
HCOMMON = $\mathcal{H}$(Common),
Sig$_A$ = $\mathcal{S}_A$(RESPCODE, AUTHTIME, HCOMMON, OPTSIG$_A$))
→ A **authorized** *value subtraction* (ID$_C$, A, 0, AUTHTIME, Ref(Common)).

4.2.5 A proving Auth(M-A)

With Auth-Request, OPTSIG$_M$, and the reconstructed Common (using values from the decrypted EncSlip as well as TID$_M$ and TID$_C$ communicated outside of *i*KP), A can prove M's authorization of *value claim*:

evidence (OPTSIG$_M$, HCOMMON, Sig$_M$, Common [TID$_M$, TID$_C$, PFLAGS, PRICE,
ID$_M$, DATE, INV-EXP, NONCE$_M$, ID$_C$, HDESC, HV, HVC]),
verify (HCOMMON = $\mathcal{H}$(Common),
Sig$_M$ = $\mathcal{S}_M$(HCOMMON, OPTSIG$_M$))
→ M **authorized** *value claim* (ID$_C$, ID$_M$, PRICE, DATE, Ref(Common)).

This rule deserves a cautionary note: if Sig$_M$ is (optionally) present in Invoice, all the evidence needed to prove S's *value claim* authorization is already present in Invoice, whereas M may not mean to commit to *value claim* already when signing the invoice. This feature is further discussed in Section 4.4.

4.2.6 M proving Auth(A-M)

With a positive Auth-Response and OPTSIG$_A$, M can prove A's authorization of *value claim*:

evidence (OPTSIG$_A$, HCOMMON, RESPCODE, AUTHTIME,
Common [TID$_M$, TID$_C$, PFLAGS, PRICE, ID$_M$, DATE,
INV-EXP, NONCE$_M$, ID$_C$, HDESC, HV, HVC]),
verify (HCOMMON = $\mathcal{H}$(Common),
RESPCODE is positive,
Sig$_A$ = $\mathcal{S}_A$(RESPCODE, AUTHTIME, HCOMMON, OPTSIG$_A$))
→ A **authorized** *value claim* (ID$_M$, A, PRICE, AUTHTIME, Ref(Common)).

For a denied authorization (negative RESPCODE), a negative Acquirer authorization (amount=0) is proved in a similar way. The conclusion becomes:
→ A **authorized** *value claim* (ID$_M$, A, 0, AUTHTIME, Ref(Common)).

4.3 Proofs of received authorizations

As discussed in Section 3.2, we want to investigate whether a signed receipt of a correctly formed *i*KP message suffices to prove that the recipient was able to verify the message (and eventually continue the protocol). The examples in *i*KP where this may apply are:

- C proving "M **received** (C **authorized** *payment* (...))": it needs to be proved that M received a valid Payment and knew all the data necessary to verify it, i.e. Common and OPTSIG$_C$. It can only be proved that M knows Common if M signed Invoice (optional). It is more difficult to prove that M knows OPTSIG$_C$, since these data are transported out-of-band. Thus, unless the mechanism transporting the out-of-band data uses signed receipts as well, *and* Sig$_M$ was part of Invoice, the statement cannot be proved.

- M proving "A **received** (M **authorized** *value claim* (...))": it needs to be proved that A received a valid Auth-Request and knew all the data necessary to verify it. This requires proving that, on top of the contents of Auth-Request, A knows TID_C, TID_M, OPTSIG_M and OPTSIG_C. Again, this can only be proved if receipt of these out-of-band data is provable.
- A similar reasoning holds for C proving "A **received** (C **authorized** *value subtraction* (...))".

As can be seen from these examples, proving that a party (M or A) received a valid *i*KP message does not prove that the party could verify the message or continue the protocol.

4.4 Discussion

4.4.1 Completeness of Authorizations

Each of the parties in 3KP obtains the necessary evidence for proving the other parties' authorizations. One discussion point is the authorizing value of Sig_M if present in Invoice, as explained in the following.

The description of the *i*KP protocols allows the Merchant to cancel a transaction (Cancel message) after receiving Payment. However, Sig_M, optionally included in Invoice, can be used by the Customer (or an intruder intercepting Payment) to compose a valid Auth-Request without awareness or consent by the Merchant. This may lead to scenarios where an honest Merchant M cancelled a payment but a dishonest party may make it seem that M authorized it at the same time.

Hence the meaning of Sig_M as proof of Auth(M-A) authorization is ambiguous. This may be solved by either:

- Keeping the protocol as is but specifying that a Merchant releasing Sig_M in Invoice immediately authorizes *value claim* and cannot issue Cancel anymore (if he does not want to be accused of cheating);
- Changing the protocol to include a different Sig_M in Auth-Request than in Invoice.

4.4.2 Revealing Data

DESC, the description of the goods, only needs to be revealed if it is part of the desired conclusion; the encrypted credit-card slip need not be revealed to the verifier to prove B's authorization of the *value subtraction*.

4.4.3 Out-of-band Data

All signatures (Sig_C, Sig_M, Sig_A) and derived signatures (V, VC) include optional data OPTSIG_Z. These data are not part of *i*KP protocol messages but are part of the evidence needed to verify or prove the implied authorization. A similar remark holds for TID_M and TID_C which are not transported as part of *i*KP messages. The inclusion of out-of-band data in the *i*KP signatures also makes it impossible to prove that the recipient of a message could verify the authorization contained in the message.

4.4.4 Signing Trees of Hashes

As shown above, all three parties sign $\mathcal{H}(\mathrm{Common})$, and their signatures commit them to all the values in $\mathcal{H}(\mathrm{Common})$. All three parties thus sign $\mathcal{H}(\mathrm{DESC}, \mathrm{SALT}_D)$. However, only C and M know DESC thus only C and M commit themselves to DESC when signing $\mathcal{H}(\mathrm{DESC}, \mathrm{SALT}_D)$: for A, signing $\mathcal{H}(\mathrm{DESC}, \mathrm{SALT}_D)$ does not entail signing its constituent parts. This distinction should be pointed out in the definition of the protocol and its provability rules.

5 SET

5.1 SET Protocol Overview

SET, like *i*KP, is based on the generic payment protocol of Figure 1. As in the *i*KP analysis, we make abstraction of SET options such as separation of authorization and clearing, or refunds.

An additional complexity of the SET protocol, as compared to *i*KP, is the possibility to split a SET payment in multiple authorization threads, and to have multiple captures (as well as credits and refunds) associated with every authorization thread. Since this 'branching out' of a SET transaction complicates the analysis, we do not consider split authorizations nor multiple captures.

A number of SET messages are encrypted using the public key of the recipient. Depending on the security requirements of the message, and depending on whether or not certain parts of the message are already in encrypted form, different types of encryption are used. From an evidence point of view, however, these different types of encryption are equivalent, since the actual evidence to an external verifier necessarily consists of the decrypted message. Therefore, we simplify the encryption notation: $\mathsf{E}_X(\mathrm{M})$ stands for a message encrypted using X's public encryption key PK_X from which X can retrieve M by decryption. In most cases, M (or a part of M) is encrypted using a symmetric (DES-) key, which is (part of) the actual data encrypted with PK_X.

For further simplification, we omit a number of fields which are either optional or not relevant to verifiability, such as thumbprints, bank identification numbers, language identifiers, request-response pair identifiers (RRPIDs), and optional extensions. Figure 4 represents the resulting, simplified, SET payment protocol.

5.2 Proofs of Authorizations

5.2.1 M proving Auth(C-M)

To prove Auth(C-M), M can use the non-encrypted part of PReq [2].

> **evidence** (OIData [TIDs, Chall_C, HOD, ODSalt, Chall_M], HPIData, PI [Sig_C, ...]),
> **verify** (HOIDATA = H(OIData),
> $\mathrm{Sig}_C = \mathrm{S}_C$(HOIDATA, HPIData))
> → C **authorized** *payment* (C, M=?, Amt=?, TIDs.PReqDate, Ref(TIDs, HOD)).

PReq alone does not prove which PurchAmt was authorized, since PurchAmt is only present in hashed (HOD) form. Thus, to prove PurchAmt, M additionally needs to provide HODContents, according to following generic extension rule:

> **evidence** (HODContents [OD, PurchAmt, ODSalt]),
> **verify** (HOD = H(OD, PurchAmt, ODSalt),
> X **authorized** *payment* (Y, Z, Amt=?, PReqDate, Ref(TIDs, HOD)))
> → X **authorized** *payment* (Y, Z, PurchAmt, PReqDate, Ref(TIDs, HOD, OD)).

By providing HODContents, M necessarily also reveals OD to V. In *i*KP, it was possible to prove Auth(C-M) of PRICE without revealing DESC.

Note that, with PReq and HODContents together, M cannot prove the payment was made to him. Only A can prove that by revealing the decrypted PI (including MID).

5.2.2 C proving Auth(M-C)

C can try to use the receipt PRes to prove Auth(M-C). PRes includes a CapRatio (ratio of captured amount CapAmt to authorized amount PurchAmt), but does not include HOD or PurchAmt. Thus, PRes alone cannot be used as a receipt for a certain CapAmt. Note, in addition, that PRes does not contain C's identity.

[2] *composite-field[component-fields]* denotes a composite field (e.g., OIData) with its individual components

- **Atomic and Composite Fields:**

E_X(M)	message M, encrypted under X's public encryption key
Signed_X(M)	message M, signed with X's private signature key (includes M)
S_X(M)	signature-only of message M with X's private signature key (does not include M)
E_X(Signed_Y(M, baggage))	simplified notation for {Sig_Y(M, baggage), E_X(M), baggage}; baggage is already encrypted and is signed together with encrypted M
TIDs	LID_C, LID_M, XID, PReqDate, Language
LID_Z	local ID of Z
Chall_Z	challenge of Z
XID	globally unique ID
HOD	H(HODContents)
HODContents	OD, PurchAmt, ODSalt
OD	Order Description
MID	Merchant ID
AuthX, X=Code,Amt,Date	code, amount, date in AuthReq
AuthResX, X=Amt, Date	amount, date in AuthRes
CompletionCode	Merchant's completion code
CapRatio	ratio of AuthAmt:PurchAmt
PANData	PAN (Private Account Number), CardExpiry, PANSecret, EXNonce
PI	S_C(HOIData, HPIData), E_A(PIHead, HOIData, PANData)
HOIData	H(OIData)
HPIData	H(PIData)
OIData	TIDs, Chall_C, HOD, ODSalt, Chall_M
PIData	PIHead, PANData
PIHead	TIDs, HOD, PurchAmt, MID

- **Protocol Flows:**

$$\begin{array}{llcl}
\text{PinitReq:} & C & \xrightarrow{\mathsf{LID}_C,\ \mathsf{LID}_M,\ \mathsf{Chall}_C} & M \\
\text{PinitRes:} & C & \xleftarrow{\mathsf{Signed}_M(\mathsf{TIDs},\ \mathsf{Chall}_C,\ \mathsf{Chall}_M)} & M \\
\text{PReq:} & C & \xrightarrow{\mathsf{OIData},\ \mathsf{HPIData},\ \mathsf{PI}} & M \\
\text{AuthReq:} & M & \xrightarrow{\mathsf{E}_A(\mathsf{Signed}_M(\mathsf{TIDs},\ \mathsf{AuthReqAmt},\ \mathsf{AuthReqDate},\ \mathsf{HOIData},\ \mathsf{HOD},\ \mathsf{PI}))} & A \\
\text{AuthRes:} & M & \xleftarrow{\mathsf{E}_M(\mathsf{Signed}_A(\mathsf{TIDs},\ \mathsf{AuthCode},\ \mathsf{AuthAmt},\ \mathsf{AuthDate}))} & A \\
\text{PRes:} & C & \xleftarrow{\mathsf{Signed}_M(\mathsf{TIDs},\ \mathsf{Chall}_C,\ \mathsf{CompletionCode},\ \mathsf{AuthCode},\ \mathsf{AuthDate},\ \mathsf{CapRatio})} & M
\end{array}$$

Figure 4: SET: Payment with in-line authorization

evidence [a](Signed$_M$(TIDs, Chall$_C$, CompletionCode, AuthCode, AuthDate, CapRatio))
→ M **authorized** *payment* (C=?, M, Amt=?, AuthDate, Ref(TIDs, CapRatio)).

[a]**evidence** (Signed$_X$(M)) is used as a short form for: **evidence** (Sig$_X$, M), **verify** (Sig$_X$ = S$_X$(M)).

What if C additionally provides PReq and HODContents to V, proving C's identity and PurchAmt for this TIDs? Unfortunately, this does not exclude a dishonest C from making up a PReq on the spot containing PurchAmt and identity C wants to prove. Or, from a PReq proving

C **authorized** *payment* (C, M, PurchAmt, AuthDate, Ref(TIDs))

and a PRes proving

M **authorized** *payment* (C=?, M, Amt=?, AuthDate, Ref(TIDs, CapRatio))

V cannot conclude

M **authorized** *payment* (C, M, PurchAmt *CapRatio, AuthDate, Ref(TIDs)))

unless C can prove to V that it was indeed that specific PReq (C and PurchAmt) M replied to. C cannot do so without cooperation from M (signing additional statements about the PReq it witnessed), or from A, providing AuthReq-AuthRes data allowing to formulate a replacement receipt:

evidence (HODContents (OD, PurchAmt, ODSalt),
Signed$_M$(TIDs, AuthReqAmt, AuthReqDate, HOIData, HOD, PI),
Signed$_A$(TIDs, AuthCode, AuthAmt, AuthDate))
→ M **authorized** *payment* (C=?, M, AuthAmt, AuthDate, Ref(TIDs, HOD)).

5.2.3 A proving Auth(C-A)

To prove authorization by C, A can use S$_C$(HOIData, HPIData) in AuthReq. Since A, without explicit cooperation from M, does not possess OIData, A tries to prove that HPIData = H(PIData), with PIData mapping the desired authorization data. This assumes revealing PIData, and thus the cleartext PANData to the verifier:

evidence (Sig$_C$, PIHead [TIDs, HOD, PurchAmt, MID], HOIData,
PANData [PAN, CardExpiry, PANSecret, EXNonce]),
verify (HPIDATA = H(PIHead, PANData),
Sig$_C$ = S$_C$(HOIData, HPIData))
→ C **authorized** *value subtraction* (C, A=?, PurchAmt, TIDs.PReqDate, Ref(TIDs, HOD)).

Revealing PANData can only be avoided if A is able to obtain OIData and HODContents (e.g., from M). In that case, A can prove Customer authorization of *value subtraction* using the same data as the Merchant uses to prove Customer authorization of *payment*:

evidence (OIData [TIDs, Chall$_C$, HOD, ODSalt, Chall$_M$], HPIData,
Sig$_C$, HODContents [OD, PurchAmt, ODSalt]),
verify (Sig$_C$ = S$_C$(HOIData, HPIData),
HOD = H(OD, PurchAmt, ODSalt))
→ C **authorized** *value subtraction* (C, A=?, PurchAmt, TIDs.PReqDate, Ref(TIDs, HOD, OD)).

5.2.4 C proving Auth(A-C)

C never receives proof of A's authorization. C can prove *value subtraction* only when Acquirer or Merchant cooperate and the verifier accepts replacement receipts, e.g., accepts an Auth-Response (targeted at M) as evidence of *value subtraction*. To assemble this evidence, C of course needs A or M to cooperate.

> **evidence** (Signed_M(AuthReqAmt, HOIData, HOD, Sig_C),
> Signed_A(TIDs, AuthCode, AuthAmt), HPIData),
> **verify** ($\mathsf{Sig}_C = \mathsf{S}_C$(HOIData, HPIData))
> $\rightarrow$ A **authorized** *value subtraction* (C, A, AuthAmt, AuthDate, Ref(TIDs, HOD)).

Note that the verification of Sig_C is needed to show that A knew the *value subtraction* came from C.

5.2.5 A proving Auth(M-A)

The contents of AuthReq allow A to prove M's authorization of *value claim*:

> **evidence** (Signed_M(TIDs, AuthReqAmt, AuthReqDate, HOIData, HOD, PI))
> $\rightarrow$ M **authorized** *value claim* (M, A=?, AuthReqAmt, AuthReqDate, Ref(TIDs, HOD)).

A cannot prove the *value claim* was made to him. This is not critical since the relationship between M and A may be derived from a contract. In addition, since AuthReq is originally encrypted with PK_A, the (decrypted) evidence proves that AuthReq at least was decrypted by the Acquirer for whom M encrypted the message.

5.2.6 M proving Auth(A-M)

The contents of AuthRes prove A's authorization of *value claim* for AuthAmt.

> **evidence** (Signed_A(TIDs, AuthCode, AuthAmt, AuthDate))
> $\rightarrow$ A **authorized** *value claim* (M=?, A, AuthAmt, AuthDate, Ref(TIDs)).

Also here, the decrypted evidence shows that AuthRes was decrypted by the Merchant for whom A encrypted the message.

5.3 Proofs of received authorizations

Also in SET, proving receipt of a valid protocol message does not prove the recipient had enough information to continue the protocol:

- Even if it can be proved that M issued a valid PinitRes and received a valid and corresponding PReq, it cannot be proved that M knew the necessary data (PurchAmt and ODSalt) to verify PReq. Even if M issues a negative PRes, it cannot be proved that M had (or agreed with) the necessary data to authorize the payment (since PRes does not contain HOD).
- If it can be proved that A received a valid AuthReq, one cannot prove that PIData was acceptable to A unless C wants to cooperate by revealing all the contents of PIData (including PANData).

5.4 Discussion

5.4.1 Completeness of Authorizations

Some of the authorizations in SET are incomplete (some of the parameters cannot be derived by the verifier) or absent (Auth(A-C)).

The absence of Auth(A-C) is important especially because also PRes is not a complete receipt. Thus, the Customer has secure receipts from neither Merchant nor Acquirer.

One may argue that PRes, in combination with PReq, may be treated by V as a valid receipt, and that cheating by C (by providing a false PReq) is likely to be detected. However, it remains an important point that, in the absence of other evidence provided by M or A, a Customer providing PRes in combination with a false PReq may cheat a verifier.

The example of PRes shows us the following: PRes may be considered by C to be a secure authorization and receipt of the amount captured. This is because C knows which PurchAmt it authorized in PReq, and thus which PurchAmt M is able to prove. However, C cannot convince V of the same fact, since PRes is not self-contained, i.e. it does not include the values that C wants to prove to V.

The consequences of these missing authorizations depend on the default behavior of a verifier presented by C with a (PReq, PRes) pair as proof of a payment. Consider a company optimizing its expense account processing by accepting electronic (SET) receipts from its employees for business-related expenses. The company's accounting department can take three different approaches to verifying these receipts:

- Since current SET implementations do not come with the necessary mechanisms to verify transaction records after the fact, the company simply trusts the employee to claim the correct amount, and ignores the actual SET transaction record. This obviously allows errors and fraudulent claims.
- The accounting department decides to invest in verification software that extracts SET receipts (in this case, PRes and PReq) from the employees' transaction records and verifies them. Still, this leaves opportunities for fraud by employees which are able to tamper with their SET software and make it produce different PReq's for the same TIDs. Suppose an employee buys an airline ticket for US$ 200 (PReq with PurchAmt = 200, PRes with CapRatio = 1). The employee then produces another PReq for the same TIDs, but with PurchAmt = 1000. With the second PReq and the PRes, the employee now proves to the accounting department's verifier that she bought a ticket for US$ 1000.
- The accounting department can, for every claim, ask for additional statements or SET evidence from the bank. But this approach is costly and certainly does not fulfill the original goal of optimization.

The example above shows how a dishonest Customer can cheat a verifier, if the verifier accepts incomplete proof. The lacking proof of authorization can of course work in the Customer's disadvantage as well. Consider a Customer who pays US$100 for a certain item described in OD, and considers the receipt of PRes as a valid proof of payment for this item. If the goods delivered do not match the negotiated terms (OD), or the customer's account is debited with another amount than expected, the Customer has to rely on additional evidence provided by the Acquirer or the Merchant. This may be a costly procedure; or, it may turn out to be impossible if the Merchant's or Acquirer's payment systems do not support the dispute handling mechanisms required for collecting this evidence.

5.4.2 Revealing Data

Since the goods description OD is hashed together with PurchAmt in HOD, neither C nor M can prove authorization of *payment* without revealing OD to the verifier.

More data than necessary is also revealed when A needs to prove Auth(C-A): to prove customer's authorization of *value subtraction* with only the evidence in its possession, the Acquirer needs to reveal the whole contents of PIData, including PANData, to the verifier.

The above problem in SET could be avoided by providing an extra level of hashing, such that only the hash H(PANData) needs to be revealed together with PIHead in order for a verifier to be able to re-compose HPIData (and thus verify PI without needing PANData).

6 Summary

In this paper, we formulated a number of provability requirements for payment protocols. Our starting assumption was that every honest participant should collect enough evidence to prove its transaction views without relying on evidence provided by other parties. First of all, this is needed to use digital receipts as self-contained pieces of evidence in off-line verification scenarios. Furthermore, relying on other parties to give evidence requires trust in those parties. Also, it assumes that these other parties' payment systems are enhanced with interoperable dispute handling and evidence collection mechanisms. This is realistic only if those mechanism are standardized to the same level as the payment system itself, and if they become a mandatory part of it.

We validated the "3KP" versions (all parties digitally signing messages) of *i*KP and SET protocols against these provability requirements, by trying to formulate rules used by a verifier to derive statements about authorizations. Both *i*KP and SET transport some protocol-related data out-of-band. This makes it difficult to 'accuse' a party of not continuing the protocol, even if this party can be proved to have received the preceding protocol message. More important, however, is the possibility to use *i*KP and SET evidence and signatures to prove parties' transaction authorizations. In this respect, we find *i*KP to provide more complete evidence than SET.

Some of the findings lead to general recommendations for constructing provable payment protocols. One example is the use of hash trees, constructed such that sensitive data can be revealed selectively. Another example is the specification of the witnessing value of a signature, e.g., with respect to the constituent data of signed hashes.

Another, more important, general finding is the following: secure authorization using digital signatures does not guarantee provability of the authorization to an external verifier, unless the message is sufficiently self-contained.

In conclusion, we recommend that protocols claiming non-repudiation explicitly specify the rules to be used for deriving authorization statements from digitally signed messages.

References

[AHS98] N. Asokan, Els Van Herreweghen, and Michael Steiner. Towards a framework for handling disputes in payment systems. In *Third USENIX Workshop on Electronic Commerce*, pages 187–202, Boston, Mass., September 1998. USENIX. Available from `http://www.zurich.ibm.com/Technology/Security/publications/1998/AvHS98b%.ps.gz`.

[APASW98] J. L. Abad-Peiro, N. Asokan, Michael Steiner, and Michael Waidner. Designing a generic payment service. *IBM Systems Journal*, 37(1):72–88, January 1998.

[ASW97] N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In Tsutomu Matsumoto, editor, *4th ACM Conference on Computer and Communications Security*, pages 6, 8–17, Zurich, Switzerland, April 1997. ACM Press.

[BGH+95] Mihir Bellare, Juan Garay, Ralf Hauser, Amir Herzberg, Hugo Krawczyk, Michael Steiner, Gene Tsudik, and Michael Waidner. iKP – a family of secure electronic payment protocols. In *First USENIX Workshop on Electronic Commerce*, pages 89–106, New York, July 1995. USENIX.

[BGH+99] Mihir Bellare, Juan Garay, Ralf Hauser, Amir Herzberg, Hugo Krawczyk, Michael Steiner, Gene Tsudik, Els Van Herreweghen, and Michael Waidner. Design, implementation and deployment of a secure account-based electronic payment system. Research Report RZ 3137, IBM Research Division, June 1999. Submitted to IEEE JSAC Special Issue on Computer Security.

[Bol97] Dominique Bolignano. Towards the formal verification of electronic commerce protocols. In *10th IEEE Computer Security Foundations Workshop*, pages 133–146. IEEE Computer Press, 1997.

[Bra97] S. Brackin. Automatic formal analyses of two large commercial protocols. In *DIMACS Workshop on Design and Formal Verification of Security Protocols*, Rutgers New Jersey, September 1997.

[Kai95] Rajashekar Kailar. Reasoning about accountability in protocols for electronic commerce. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May 1995. IEEE Computer Society Press.

[Kat96] M. Ethan Katsch. Dispute resolution in cyberspace. In *Connecticut Law Review Symposium: Legal Regulation of the Internet*, number 953 in 28, 1996. Available from `http://www.umass.edu/legal/articles/uconn.html`.

[KN98] Volker Kessler and Heike Neumann. A sound logic for analysing electronic commerce protocols. In J.-J. Quisquater, Y. Deswarte, C. Meadows, and D. Gollmann, editors, *Proceedings of the Fifth European Symposium on Research in Computer Security (ESORICS)*, number 1485 in Lecture Notes in Computer Science, Louvain-la-Neuve, Belgium, September 1998. Springer-Verlag, Berlin Germany.

[MS98] Catherine Meadows and Paul Syverson. A formal specification of requirements for payment transactions in the SET protocol. In *Proceedings of the Financial Cryptography Conference (FC98)*, 1998.

[MV97] Mastercard and Visa. *SET Secure Electronic Transactions Protocol*, version 1.0 edition, May 1997. Book One: Business Specifications, Book Two: Technical Specification, Book Three: Formal Protocol Definition. Available from http://www.setco.org/set_specifications.html.