
Michael Backes
IBM Zurich Research Laboratory
Rüschlikon, Switzerland
mbc@zurich.ibm.com

Markus Dürmuth†
IAKS, Arbeitsgruppe Systemssicherheit,
Karlsruhe, Germany
markus.duermuth@web.de

October 21, 2006

Abstract

We present the first cryptographically sound Dolev-Yao-style security proof of a comprehensive electronic payment system. The payment system is a slightly simplified variant of the 3KP payment system and comprises a variety of different security requirements ranging from basic ones like the impossibility of unauthorized payments to more sophisticated properties like disputability. We show that the payment system is secure against arbitrary active attacks, including arbitrary concurrent protocol runs and arbitrary manipulation of bitstrings within polynomial time if the protocol is implemented using provably secure cryptographic primitives. Although we achieve security under cryptographic definitions, our proof does not have to deal with probabilistic aspects of cryptography and is hence within the scope of current proof tools. The reason is that we exploit a recently proposed Dolev-Yao-style cryptographic library with a provably secure cryptographic implementation. Together with composition and preservation theorems of the underlying model, this allows us to perform the actual proof effort in a deterministic setting corresponding to a slightly extended Dolev-Yao model.

1 Introduction

It is hardly necessary today to justify or stress the importance of electronic commerce, which has been rapidly gaining momentum since the early nineties, and is equally appealing to online merchants, consumers, and payment providers. The core of electronic commerce is an electronic payment system that is supposed to fulfill the individual requirements of the participating parties. These range from standard requirements like the impossibility of unauthorized payments, to more sophisticated ones like granting individuals the ability to succeed in disputes in cases where they have been betrayed. Devising a payment system that lives up to these requirements has been a challenging task, and many payment systems that were claimed to be provably secure have fallen prey to subsequent attacks in the past [74, 73]. Today, it is commonly agreed that cryptographic protocols in general and payment systems in particular have to contain a rigorous proof of security in order to be acceptable.

One way to conduct such a proof is the cryptographic approach. Its security definitions are based on complexity theory, e.g., [53, 51, 35]. The security of a cryptographic protocol is proved by reduction,

---

*An earlier version of this work appeared at [7].
†This work was conducted while the second author was affiliated with the IBM Zurich Research Laboratory.
i.e., by showing that breaking the protocol implies breaking one of the underlying cryptographic primitives with respect to its cryptographic definition and thus finally a computational assumption such as the hardness of integer factoring. This approach captures a very comprehensive adversary model and allows mathematically rigorous proofs. However, because of probabilism and computational restrictions, these proofs have had to be done by hand so far, which often yields proofs with faults or gaps. Moreover, such proofs rapidly become too complex for larger protocols, which was one of the main reasons why even comparatively small payment systems have proved considerably error-prone in the past.

The alternative is the formal-methods approach, which is concerned with the automation of proofs using model checkers and theorem provers. As these tools currently cannot deal with cryptographic details such as error probabilities and computational restrictions, abstractions of cryptography are used. They are almost always based on the so-called Dolev-Yao model [49], which represents cryptography as term algebras. The use of term algebras simplifies proofs of larger protocols considerably and has led to a large body of literature on analyzing protocol security using various techniques for formal verification, e.g., [68, 64, 59, 38, 72, 1].

Employing the Dolev-Yao abstraction—or abstractions of a similar flavor—to the analysis of a payment system using tool support or paper-based reasoning has proved to be an extremely valuable approach; a far from exhaustive list of work along those lines includes [58, 37, 36, 60, 65, 31, 32]. Although these approaches are suitable for reasoning about the security of large-scale systems, their drawback is that they exist only in the Dolev-Yao model and there is no theorem that carries these results over to the cryptographic approach with its much more comprehensive adversary.

We close this gap by providing the first security proof of a payment system that is both within the scope of formal proof tools and is sound with respect to the rigorous definitions and the comprehensive adversary model of cryptography. The payment system is a slightly simplified variant of the 3KP payment system [34, 33] and comprises a variety of different security requirements ranging from basic ones like the impossibility of unauthorized payments and weak atomicity to more sophisticated properties like disputability. More precisely, we show that the payment system is secure against arbitrary active attacks, including arbitrary concurrent protocol runs and arbitrary manipulation of bitstrings within polynomial time. The underlying model ensures strong composability so that our payment system can be used as a submodule within larger protocols without degrading its proved security properties. The underlying assumption is that the Dolev-Yao-style abstraction of digital signatures is implemented using a chosen-message secure digital signature scheme with small additions like signature tagging. Chosen-message security was introduced in [54], and efficient signature systems that are secure in this sense exist under reasonable assumptions [54, 48, 50].

Our proof relies on a recent general result that a so-called ideal cryptographic library, which implements a slightly extended Dolev-Yao model, can be securely realized by a specific cryptographic implementation [22, 24, 16, 28]. A composition theorem for the underlying security notion implies that protocol proofs can be made using the ideal library, and security then carries over automatically to the cryptographic realization. However, because of the extension to the Dolev-Yao model, no prior formal-methods proof carries over directly. Besides its value for the analysis of electronic payment systems, the proof shows that, in spite of the extensions and differences in presentation with respect to prior Dolev-Yao models, a proof can be made over the new library that seems easily accessible to current automated proof tools. In particular, the proof contains neither probabilism nor computational restrictions.

**Related Work.** The design of electronic payment systems has a long history, dating back to the eighties and early nineties [43, 44, 47, 45, 46, 71]. Based on these works, a substantial body of commercial research has emerged, focusing on various aspects of payment systems, from protocol design and implementation to security analysis and formal verification. The Dolev-Yao model has played a central role in this work, as it provides a framework for reasoning about the security of protocols in a comprehensible and mathematically rigorous way. However, as the complexity of these systems has increased, so has the need for formal methods to ensure their security, leading to the development of formal-methods approaches.

1Efforts exist to formulate syntactic calculi for dealing with probabilism and polynomial-time considerations, in particular [69, 70, 57]. However, this approach cannot yet handle protocols with any degree of automation.
attempts at electronic payment systems emerged. The iKP family [34, 33] constituted one of the most important of those attempts. It is the direct predecessor of today’s prevailing SET standard, and offered a variety of strong security guarantees while still relying on relatively simple underlying mechanisms. We refer to [4] for an exhaustive overview of the other attempts.

Work on justifying Dolev-Yao-style models under cryptographic definitions prior to [22] was restricted to passive adversaries and symmetric encryption [3, 2, 61]. Concurrently with [22], an extension to asymmetric encryption—but still under passive attacks only—was presented in [56]. The underlying Master’s thesis [55] considers asymmetric encryption under active attacks, but does so in the random oracle model, which is itself an idealization of cryptography and is not justifiable [41]. The recent work of [67] gives a slightly more efficient implementation of asymmetric encryption than [22] (no additional tagging and randomization) at the cost of a much less general library and a weaker security notion. The outlook in [67] would essentially yield [22] again. Since computational soundness has become a highly active line of research, we exemplarily list further recent results in this area without going into further details [11, 23, 26, 20, 9, 17, 8, 29, 12].

Based on the ideal Dolev-Yao-style library, the well-known Needham-Schroeder-Lowe, Otway-Rees and Yahalom protocols have been analyzed in a cryptographically sound way [15, 5, 19]. In contrast to the proof in this paper, these proofs did not have to reason about digital signatures and related aspects like non-repudiation, and the protocols are rather small examples compared to a comprehensive payment system. We stress though that our result serves as further exemplification of the usefulness of the cryptographic library [22], their extensions [16, 28], and the corresponding general theorems for linking symbolic and cryptographic properties based on this library [10, 21, 13, 14, 18, 6] for the cryptographically sound verification of cryptographic protocols. Recently, [42] showed how to translate a specific class of protocols expressed in the UC framework [40] into corresponding representations in the strand space approach and plans to exploit existing tools for this specific Dolev-Yao model, e.g., Athena [77]. At the moment, the work is restricted to mutual authentication and key exchange protocols, and in contrast to [22] hence does not address soundness of Dolev-Yao models in their usual generality.

The security notion used for the relation between the ideal Dolev-Yao-style library and its cryptographic implementation, reactive simulatability, and its composition properties were introduced in [75] and extended to asynchronous systems in [76, 40, 27, 25]. It extends the security notions of multiparty (one-step) function evaluation [78, 51, 52, 66, 30, 39] and the observational equivalence of [62]. There are multiple possible layers of sound abstraction from cryptography in the sense of reactive simulatability besides Dolev-Yao-style cryptographic libraries. They reach from low-level idealizations that still have real cryptographic in- and outputs to high-level abstractions like secure channels. The specific aspects of a Dolev-Yao-style abstraction are simple operator-tree abstractions from nested cryptographic terms, the restriction of adversary capabilities to algebraic operations on such terms, and the assumption that terms whose equality cannot be derived explicitly are always unequal.

2 Description of the Payment Protocol

Let $u$ be a client, $v$ a merchant, and $ac$ the acquirer. We assume that $u$, $v$, and $ac$ initially agreed on a description $d$ of the good and its price $p$. A successful termination of the protocol will then ensure that the parties used the same description and the same price as their local inputs to the protocol, i.e., a party can neither cheat by tampering with these inputs nor can it re-exploit an already finished payment transaction. To simplify notation we let signatures include the signed message. We further assume that every participating party $w$ initially holds a secret signature key $sk_w$ and that the corresponding public key has already been distributed authentically to the other parties.

Figure 1 summarizes the main, so-called optimistic part of the protocol in the usual protocol notation.
The part between the dotted lines contains the description of the actual protocol, consisting of five steps executed among client \( u \), merchant \( v \), and acquirer \( \text{ac} \). The parts above and below the dotted lines represent the local inputs and outputs of the protocol, respectively. They correspond to interface events that enable interaction with the users of the payment system or with higher-level programs. The protocol belongs to the class of pay-now protocols which have in common that inputs pay, receive, and allow from the client, the merchant, and the acquirer, respectively, and the outputs paid, received, and transfer to the client, the merchant, and the acquirer, respectively, occur in one single transaction. Besides its optimistic part, the protocol further offers a separate dispute part, which allows each party to contact a trusted third party to resolve disputes. We will elaborate on both parts of the protocol in the following.

**Optimistic Part.** The merchant \( v \) starts the protocol upon receiving a local input \((\text{receive}, d, p, u)\), which indicates agreement to receive the money \( p \) in exchange for the good \( d \) from \( u \). The merchant computes a signature \( \text{sig}_v := \text{sign}_\text{sk}_v(\text{invoice}, d, p, u, v) \) and sends \((\text{invoice}, \text{sig}_v)\) to client \( u \).

Upon receiving a message \((\text{invoice}, \text{sig}_v)\), the client \( u \) tests if \( \text{sig}_v \) is a valid signature with respect to \( v \)’s public key of correctly formed data. If \( u \) has not received a local command \((\text{pay}, d, p, v)\), which authorizes this payment, he stores the received data \( d, p, v \) and waits for this local command. If it has already occurred or when it occurs, \( u \) computes \( \text{sig}_u := \text{sign}_\text{sk}_u(\text{payment}, d, p, u, v) \) and sends \((\text{payment}, \text{sig}_u)\) to \( v \).

Upon receiving a message \((\text{payment}, \text{sig}_u)\), the merchant \( v \) tests if \( \text{sig}_u \) is a valid signature with respect to \( u \)’s public key of the correct data. If \( v \) has sent an invoice with the same parameters \( d, p, u, v \) to client \( u \) before, he saves \( \text{sig}_u \) for later use in disputes and sends \((\text{auth}\_\text{request}, \text{sig}_u, \text{sig}_v)\) to the acquirer \( \text{ac} \).

Upon receiving \((\text{auth}\_\text{request}, \text{sig}_u, \text{sig}_v)\), the acquirer \( \text{ac} \) tests if both signatures are valid signatures with respect to the respective public keys and if the data \( d, p, u, v \) contained in both signatures are identical. If \( \text{ac} \) has not yet received a local command \((\text{allow}, d, p, u, v)\) indicating consent to the payment, he stores the received data until this local command occurs. If it has already occurred or when it occurs, \( \text{ac} \) computes \( \text{sig}_\text{ac} := \text{sign}_\text{sk}_\text{ac}(\text{auth}\_\text{response}, d, p, u, v) \), sends \((\text{auth}\_\text{response}, \text{sig}_\text{ac})\) to \( v \), stores this payment as being processed to prevent multiple deductions of money for the same payment transaction, and outputs \((\text{transfer}, d, p, u, v)\) locally.

Upon receiving \((\text{auth}\_\text{response}, \text{sig}_\text{ac})\), the merchant \( v \) checks the validity of the signature with respect to \( \text{ac} \)’s public key and if \( v \) has earlier sent an authentication request to the acquirer containing \( d, p, u, \) and \( v \). He then sends \((\text{confirm}, \text{sig}_\text{ac})\) to client \( u \) and outputs \((\text{received}, d, p, u)\) locally.

Upon receiving \((\text{confirm}, \text{sig}_\text{ac})\), the client checks the validity of the signature with respect to \( \text{ac} \)’s public key and if \( u \) has earlier sent a payment with matching data. He then outputs \((\text{paid}, d, p, v)\) locally.

**Disputes.** Disputes enable a party to prove that specific outputs have occurred. Note that the trusted third party is not involved in the optimistic part of the protocol as described above, but it will only be invoked if two parties disagree whether the payment took place or not.

The structure of the dispute protocol is very simple, hence we omit a picture along the lines of Figure 1. A party \( w \) (either a client \( u \), a merchant \( v \), or the acquirer \( \text{ac} \)) can start a dispute by sending a local command \((\text{dispute}, d, p, v)\) (or \((\text{dispute}, d, p, u, v)\) if \( w = \text{ac} \)). As a prerequisite to initiate a dispute, \( w \) must have received the signatures of the corresponding parties in the optimistic part of the protocol execution. In this case, \( w \) computes \( \text{sig}^* := \text{sign}_\text{sk}_w(\text{dispute}, \text{sig}_x, \text{sig}_{x'}) \) where \( \{x, x'\} = \{u, v, \text{ac}\} \setminus \{w\} \) and sends \( \text{sig}^* \) to the trusted third party. Upon receiving a message \( \text{sig}^* \) from \( w \), the trusted third party checks if the signature is valid for \( w \)’s public key, if it is of the correct form, and if both contained signatures are valid signatures for the respective public keys and of the correct and matching data. In this case it outputs \((\text{dispute}, \text{true}, d, p, u, v)\), and \((\text{dispute}, \text{false}, d, p, u, v)\) otherwise.
Almost all formal proof techniques for protocols first need a reformulation of the protocol into a more
detailed version than the five steps above. These details include necessary tests on received messages,
the types and generation rules for values like $u$ and $\text{sig}_u$, and a surrounding framework specifying the
number of participants, the possibilities of multiple protocol runs, and the adversary capabilities. The
same is true when using the Dolev-Yao-style cryptographic library from [22], i.e., it plays a similar role
in our proof as “the CSP Dolev-Yao model” or “the inductive-approach Dolev-Yao model” in other proofs.
Our protocol formulation in this framework is given in Section 3.1.2 We there explain this formulation in
detail exemplarily for the clients, and then explain general aspects of the surrounding framework as far
as needed in our proofs.

3 Detailed Protocol Descriptions

We write “:=” for deterministic assignment, and $\downarrow$ is an error element available as an addition to the
domains and ranges of all functions and algorithms. The framework is automata-based, i.e., protocols are
executed by interacting machines, and event-based, i.e., machines react on received inputs. We assume a
set $M := \{1, \ldots, n\}$ of users that is partitioned into a set $M^{\text{client}}$ of clients, a set $M^{\text{merchant}}$ of merchants,
an acquirer $\text{ac}$, and a trusted third party $\text{ttp}$. By $M^{\text{PS}}_u$ we denote the payment protocol machine for a user
$u$. Let $\Sigma$ denote a finite alphabet and let $\Sigma^*$ denote the set of strings over it.

For some frameworks there are compilers to generate these detailed protocol descriptions, e.g., [63]. This should be possible
for this framework in a similar way.
3.1.1 Clients

Let \( u \in \mathcal{M}_{\text{client}} \) denote a client. The main data structure of \( M^\text{PS}_u \) is a database \( D^\text{PS}_u \) for storing the initial information related to the payments, their current status, as well as additional information gained during the protocol execution. Generally, a database \( D \) is a set of functions, called entries, each over a finite domain called attributes. For an entry \( x \in D \), the value at an attribute \( att \) is written \( x.att \). For a predicate \( \text{pred} \) involving attributes, \( D[\text{pred}] \) means the subset of entries whose attributes fulfill \( \text{pred} \). If \( D[\text{pred}] \) contains only one element, we use the same notation for this element. Adding an entry \( x \) to \( D \) is abbreviated \( D := x \). Further, we write the list operation as \( l := (x_1, \ldots, x_j) \), and the arguments are unambiguously retrievable as \( l[i] \), with \( l[i] = \bot \) if \( i > j \). In our case, each entry \( x \) in \( D^\text{PS}_u \) can have the arguments

\[
(\text{ind}, \text{desc}, \text{price}, \text{merch}, \text{sig}_m, \text{sig}_ac, \text{status}).
\]

where the arguments have the following types and meaning:

- \( x.\text{ind} \in \mathcal{IHND}S \), called index, consecutively numbers all entries in \( D^\text{PS}_u \). The set \( \mathcal{IHND}S \) is isomorphic to \( \mathbb{N} \) and is used to distinguish index arguments from others. The index is used as a primary key attribute of the database, i.e., we write \( D^\text{PS}_u[i] \) for the selection \( D^\text{PS}_u[\text{ind} = i] \). We further use the convention that look-ups in \( D^\text{PS}_u \) always return the element with the smallest index whose attributes fulfill the queried predicate.

- \( x.\text{desc} \in \Sigma^* \) is the description of the good to be purchased.

- \( x.\text{price} \in \mathbb{N} \) denotes the price of the good.

- \( x.\text{merch} \in \mathcal{M}_{\text{merchant}} \) is the identifier of the merchant that should receive the payment.

- \( x.\text{sig}_m, x.\text{sig}_ac \in \mathcal{HND}S \) denote handles to the merchant’s and the acquirer’s signature, respectively. They will be stored during the execution of the protocol and read only for disputes. The set \( \mathcal{HND}S \) is yet another set isomorphic to \( \mathbb{N} \). We always use a superscript “hnd” for handles.

- \( x.\text{status} \in \{\text{invoice}, \text{pay}, \text{processed}\} \) denotes the status of the transaction. Here invoice means that the client has received the invoice of the merchant, pay that the client gave consent to the payment, and processed that both events happened and that the payment has hence been performed.

Initially, \( D^\text{PS}_u \) is empty. \( M^\text{PS}_u \) furthermore a variable \( \text{cur}\_\text{ind}_u \in \mathcal{IHND}S \) initialized with 0 counting the size of \( D^\text{PS}_u \), and used as index for new entries in \( D^\text{PS}_u \).

The first type of input that \( M^\text{PS}_u \) can receive is a message \( (\text{pay}, d, p, v) \) from its user denoting that consent for a payment with description \( d \), price \( p \), and merchant \( v \) is given. User inputs are distinguished from network inputs by arriving at a so-called port \( \text{PS}\_\text{in}_u \). The “?” for input ports follows the CSP-convention, and “PS” stands for payment system because the user interface is the same for all payment system of the considered kind. The reaction on this input is described in Algorithm 1. \( M^\text{PS}_u \) first checks if a corresponding invoice with the same parameters has already been received before. In this case, the machine \( M^\text{PS}_u \) declares this entry to be processed and builds up a term corresponding to the payment message of the protocol using the ideal cryptographic library. The command store inserts arbitrary application data into the cryptographic library. The command list forms a list and sign creates an abstract digital signature entry. The final command send means that \( M^\text{PS}_u \) attempts to send the resulting term to \( v \) over an insecure channel. If no prior invoice message with suitable parameters occurred, \( M^\text{PS}_u \) only creates a new database entry that will be processed when the invoice message is received. The superscript \( \text{hnd} \) on most parameters denotes that these are handles, i.e., local names that this machine has for the corresponding terms. This is an important aspect of [22] because it allows the same protocol description
to be implemented once with Dolev-Yao-style idealized cryptography and once with real cryptography. More precisely, the four commands we saw so far and their input and output domains belong to the interface (in the same sense as, e.g., a Java interface) of the underlying cryptographic library. This interface is implemented by both the idealized and the real version. In the first case, the handles are local names of Dolev-Yao-style terms, in the second case of real cryptographic bitstrings. We say more about these two implementations below. The effect of send_i in the ideal implementation is that the adversary obtains a handle to the Dolev-Yao-style term and can decide what to do with it (such as forwarding it to $M_u^{PS}$ or performing Dolev-Yao-style algebraic operations on the term); the effect in the real implementation is that the adversary obtains the real bitstring and can perform arbitrary bit manipulations on it.

The behavior of $M_u^{PS}$ upon receiving an input from the cryptographic library (corresponding to a message that arrives over the network) is defined similarly in Algorithm 2. The input arrives at port out$_u$, and is of the form $(v, u, i, l^{\text{hnd}})$ where $v$ is the supposed sender, $i$ denotes that the channel is insecure, and $l^{\text{hnd}}$ is a handle to a list. The port out$_u$ is connected to the cryptographic library, whose two implementations represent the obtained Dolev-Yao-style term or real bitstring, respectively, to the protocol in a unified way by a handle. In this algorithm, $M_u^{PS}$ first determines if the message corresponds to an invoice message or a confirmation message, i.e., if the message could correspond to the first or the fifth message in the protocol description. In the first case, $M_u^{PS}$ first determines if the contained signature is a valid signature of the merchant for the correct data, and aborts at failure. If the user already gave consent to the payment, i.e., if a suitable entry with status pay already exists in the database, $M_u^{PS}$ stores the merchant’s signature, updates the status of this payment to processed, constructs a message according to the protocol description, and sends it to the intended recipient. If payment consent has not been given yet, $M_u^{PS}$ only creates a new database entry that contains the payment information of the invoice message. The evaluation of a confirmation message works similarly: $M_u^{PS}$ checks the validity of the acquirer’s signature and if a suitable entry already exists in the database, and in that case signals to its user at port PS$_\text{out}_u$ the successful completion of the payment.

Finally, $M_u^{PS}$ can receive a dispute message $(\text{dispute}, d, p, v)$ from its user at PS$_\text{in}_u$?. The behaviour of $M_u^{PS}$ on this input is defined in Algorithm 3. It first checks if an entry with corresponding parameters already exists in the database. If furthermore the corresponding signatures of the respective merchant and the acquirer have already been received for this entry, $M_u^{PS}$ builds up a term according to the protocol
Algorithm 2 Client: Evaluation of Inputs from the Cryptographic Library in \( M^\text{PS}_u \)

Input: \((v, u, i, l^{\text{hnd}})\) at \(\text{out}_u\) with \(v \in \mathcal{M}_{\text{merchant}}\).

1. \(l_j^{\text{hnd}} \leftarrow \text{list proj}(l^{\text{hnd}}, j)\) for \(j = 1, 2\).
2. \(l_1 \leftarrow \text{retrieve}(l_1^{\text{hnd}})\).
3. If \(l_1 = \text{invoice}\) then
   4. \(m_2^{\text{hnd}} \leftarrow \text{msg of sig}(l_2^{\text{hnd}})\).
   5. \(b \leftarrow \text{verify}(l_2^{\text{hnd}}, \text{pks}_{u,v}, m_2^{\text{hnd}})\).
   6. \(x_j^{\text{hnd}} \leftarrow \text{list proj}(m_2^{\text{hnd}}, j)\) for \(j = 1, \ldots, 5\).
   7. \(x_j \leftarrow \text{retrieve}(x_j^{\text{hnd}})\) for \(j = 1, \ldots, 5\).
8. \(i := D^\text{PS}_u[\text{desc} = x_2 \land \text{price} = x_3 \land \text{merch} = x_5].\text{ind}\).
9. If \(x_1 = \text{invoice} \land x_4 = u \land x_5 = v \land b = \text{true} \land i \neq \downarrow \land D^\text{PS}_u[i].\text{status} = \text{pay}\) then
10. \(D^\text{PS}_u[i].\text{sig}_m := l_2^{\text{hnd}}\).
11. \(D^\text{PS}_u[i].\text{status} := \text{processed}\).
12. \(\text{payment}^{\text{hnd}} \leftarrow \text{store(payment)}\).
13. \(m_4^{\text{hnd}} \leftarrow \text{list(payment}^{\text{hnd}}, x_2^{\text{hnd}}, x_3^{\text{hnd}}, x_4^{\text{hnd}}, x_5^{\text{hnd}})\).
14. \(s_1^{\text{hnd}} \leftarrow \text{sign(sks}_u, m_1^{\text{hnd}})\).
15. \(m^{\text{hnd}} \leftarrow \text{list(payment}^{\text{hnd}}, s_1^{\text{hnd}})\).
16. \(\text{send}_i(v, m^{\text{hnd}})\).
17. Else if \(x_1 = \text{invoice} \land x_4 = u \land x_5 = v \land b = \text{true} \land i = \downarrow\) then
18. \(D^\text{PS}_u := (\text{cur ind}_u++, x_2, x_3, x_5, l_2, i, \downarrow, \text{invoice})\).
19. End if
20. Else if \(l_1 = \text{confirm}\) then
21. \(m_2^{\text{hnd}} \leftarrow \text{msg of sig}(l_2^{\text{hnd}})\).
22. \(b \leftarrow \text{verify}(l_2^{\text{hnd}}, \text{pks}_{u,ac}, m_2^{\text{hnd}})\).
23. \(x_j^{\text{hnd}} \leftarrow \text{list proj}(m_2^{\text{hnd}}, j)\) for \(j = 1, \ldots, 5\).
24. \(x_j \leftarrow \text{retrieve}(x_j^{\text{hnd}})\) for \(j = 1, \ldots, 5\).
25. \(i := D^\text{PS}_u[\text{desc} = x_2 \land \text{price} = x_3 \land \text{merch} = x_5 \land \text{sig}_ac = \downarrow \land \text{status} = \text{processed}].\text{ind}\).
26. If \(x_1 = \text{auth response} \land x_4 = u \land x_5 = v \land b = \text{true} \land i 
eq \downarrow\) then
27. \(D^\text{PS}_u[i].\text{sig}_ac := l_2^{\text{hnd}}\).
28. Output (paid, \(x_2, x_3, x_5\)) at \(\text{PS out}_u\).
29. End if
30. End if

Algorithm 3 Client: Evaluation of User Inputs for Disputes in \( M^\text{PS}_u \)

Input: \((\text{dispute}, d, p, v)\) at \(\text{PS ind}_u\) with \(d \in \Sigma^*, p \in \mathbb{N}, \) and \(v \in \mathcal{M}_{\text{merchant}}\).

1. If \(i := D^\text{PS}_u[\text{desc} = d \land \text{price} = p \land \text{merch} = v \land \text{sig}_m \neq \downarrow \land \text{sig}_ac \neq \downarrow].\text{ind} \neq \downarrow\) then
2. \(\text{dispute}^{\text{hnd}} \leftarrow \text{store(dispute)}\).
3. \(l^{\text{hnd}} \leftarrow \text{list(dispute}^{\text{hnd}}, D^\text{PS}_u[i].\text{sig}_m, D^\text{PS}_u[i].\text{sig}_ac)\).
4. \(s^{\text{hnd}} \leftarrow \text{sign(sks}_u, l^{\text{hnd}})\).
5. \(m^{\text{hnd}} \leftarrow \text{list}(s^{\text{hnd}})\).
6. \(\text{send}_i(ttp, m^{\text{hnd}})\).
7. End if
description and sends it to the trusted third party.

Every algorithm should immediately abort the handling of the current input if a cryptographic command does not yield the desired result, e.g., if a database look-up fails. For readability we omitted this in the algorithmic descriptions; instead we impose the following convention on these and the following algorithms.

**Convention 1** If $M^P_u$ for $u \in \mathcal{M}$ receives $\downarrow$ as the answer of the cryptographic library to a command, then $M^P_u$ aborts the execution of the current algorithm, except for the command types list, proj or send.$i$.

### 3.1.2 Merchants

Let $u \in \mathcal{M}_{\text{merchant}}$ denote a merchant. Similar to the protocol machines of the clients, the machine $M^P_u$ maintains an initially empty database $D^P_u$ as its main data structure, together with a variable $\text{cur}_{\text{ind}}^u \in I\mathcal{N}DS$ for counting the size of the database and creating new indices. The entries of $D^P_u$ have the form

$$(\text{ind}, \text{desc}, \text{price}, \text{client}, \text{sig}_c, \text{sig}_{ac}).$$

For $x \in D^P_u$:

- $x.\text{ind}, x.\text{desc}, x.\text{price},$ and $x.\text{sig}_{ac}$ are defined as in the database of the client machines.
- $x.\text{client} \in \mathcal{M}_{\text{client}}$ denotes the client of this transaction.
- $x.\text{sig}_c \in \mathcal{H}\mathcal{N}DS$ denotes the handle of the client’s signature, which will be collected during the protocol execution.

The machine $M^P_u$ accepts inputs for sending an invoice to a client and for initiating a dispute from its user at port $\text{PS}_{\text{in}}^u?$, and inputs from the cryptographic library at port $\text{in}^u?$. After explaining the behavior of the protocol machines of the clients, the behavior of $M^P_u$ should be essentially clear from the protocol description of Section 2. We postpone the algorithmic description to the Appendix.

### 3.1.3 Acquirer

The machine $M^P_{\text{ac}}$ of the acquirer maintains a variable $\text{cur}_{\text{ind}}^\text{ac} \in I\mathcal{N}DS$ initialized with 0 and an initially empty database $D^P_{\text{ac}}$, where each entry $x \in D^P_{\text{ac}}$ can have arguments

$$(\text{ind}, \text{desc}, \text{price}, \text{client}, \text{merch}, \text{sig}_c, \text{sig}_m, \text{status})$$

with the following types and meanings:

- $x.\text{ind}, x.\text{desc}, x.\text{price}, x.\text{client}, x.\text{merch}, x.\text{sig}_c,$ and $x.\text{sig}_m$ are defined as in the databases of the client and merchant machines.
- $x.\text{status} \in \{\text{allow}, \text{auth}\_\text{request}, \text{processed}\}$ denotes the status of the transaction, cf. the state of the client machines. Here allow means that the acquirer has given consent to the payment, auth\_request that the merchant requested authentication of the payment, and processed that both events happened.

The machine $M^P_{\text{ac}}$ accepts inputs for processing a payment from user $\text{ac}$ at port $\text{PS}_{\text{in}}^\text{ac}?$, and inputs from the cryptographic library at port $\text{in}^\text{ac}?$. The algorithmic description of $M^P_{\text{ac}}$ is postponed to the Appendix.
3.1.4 Trusted Third Party

The machine $M_{\text{trp}}^{PS}$ of the trusted third party only accepts input from the cryptographic library at port $\text{in}_{\text{trp}}$. Upon receiving such an input, it first checks the validity of the signature and if the message is indeed a well-formed dispute message. It then identifies the parties that are involved in the payment and determines whether the dispute should be allowed or denied by checking the validity of the contained signatures and if they have been issued on matching parameters. The algorithmic description can be found in the Appendix.

3.2 Further Initial State

We have assumed in the algorithms that each protocol machine $M_{u}^{PS}$ already has a handle $sk_{u,\text{hnd}}$ to its own secret signature key and handles $pk_{v,\text{hnd}}$ to the public keys of every participant $v$. The cryptographic library can also represent key generation and distribution by normal commands. Further, we assume that each machine $M_{u}^{PS}$ contains the bitstring $u$ denoting its identity.

3.3 Overall Framework and Adversarial Model

The framework that determines how machines such as our payment system machines and the machines of the idealized or real cryptographic library execute is taken from [76, 27]. The basis is an asynchronous probabilistic execution model with distributed scheduling. We already used implicitly above that for term construction and parsing commands to the cryptographic library, a so-called local scheduling is defined, i.e., a result is returned immediately. The idealized or real network sending via this library, however, is scheduled by the adversary.

When protocol machines such as $M_{u}^{PS}$ for certain users $u \in \{1, \ldots, n\}$ are defined, there is no guarantee that all these machines are correct. A trust model determines for what subsets $\mathcal{H}$ of $\{1, \ldots, n\}$ we want to guarantee anything; in our case this is for all subsets which comprise at least the trusted third party. Incorrect machines disappear and are replaced by the adversary. Each set of potential correct machines together with its user interface is called a structure, and the set of these structures is called the system. When considering the security of a structure, an arbitrary probabilistic machine $H$ is connected to the user interface to represent all users, and an arbitrary machine $A$ is connected to the remaining free ports (typically the network) and to $H$ to represent the adversary, see Figure 2. In polynomial-time security proofs, $H$ and $A$ are polynomial-time.

This setting, more precisely the underlying model of secure reactive systems [76, 27], implies that any number of concurrent protocol runs with both honest participants and the adversary are considered
because \( H \) and \( A \) can arbitrarily interleave local inputs of the payment protocol with the delivery of network messages.

For a set \( H \) of honest participants, the user interface of the ideal and real cryptographic library is the port set \( S^\text{cry}_H := \{ \text{in}_u, \text{out}_u \mid u \in H \} \). This is where the payment protocol machines input their cryptographic commands and obtain results and received messages. In the ideal case this interface is served by just one machine \( TH^\text{cry}_H \) called trusted host which essentially administrates Dolev-Yao-style terms under the handles. In the real case, the same interface is served by a set \( M_H^\text{cry} := \{ M^\text{cry}_u \mid u \in H \} \) of real cryptographic machines. The corresponding systems are called \( S_{sys}^\text{cry, id} := \{(\{ TH^\text{cry}_H \}, S^\text{cry}_H) \mid H \subseteq \{1, \ldots, n\}\} \) and \( S_{sys}^\text{cry, real} := \{(M^\text{cry}_H, S^\text{cry}_H) \mid H \subseteq \{1, \ldots, n\}\} \).

The user interface of the payment protocol machines is given by \( S^\text{Sys}_H := \{ \text{PS}_{in}u, \text{PS}_{out}u \mid u \in H \setminus \{ \text{ttp} \}\} \cup \{ \text{PS}_{out} \text{ttp} \}, \) cf. Figure 2. The ideal and real payment systems serving this interface differ only in the cryptographic library. With \( M_H^\text{Sys} := \{ M^\text{Sys}_u \mid u \in H \}, \) they are \( S_{sys}^\text{Sys, id} := \{(M^\text{Sys}_H \cup \{ TH^\text{cry}_H \}, S^\text{Sys}_H) \mid \{ \text{ttp} \}\subseteq \{1, \ldots, n\}\} \) and \( S_{sys}^\text{Sys, real} := \{(M^\text{Sys}_H \cup M^\text{cry}_H, S^\text{Sys}_H) \mid \{ \text{ttp} \}\subseteq \{1, \ldots, n\}\} \).

### 3.4 On Polynomial Runtime

In order to be valid users of the real cryptographic library, the machines \( M^\text{Sys}_u \) have to be polynomial-time. We therefore define that each machine \( M^\text{Sys}_u \) maintains explicit polynomial bounds on the accepted message lengths and the number of inputs accepted at each port. As this is done exactly as in the cryptographic library, we omit the rigorous write-up.

### 4 The Security Properties

The arguably most important security property of a payment system is that no money can be transferred without the client's consent. This can be captured as an integrity property \( \text{Req} \) contains the "good" traces at these ports. A trace is a sequence of sets of events. We write an event \( p?m \) or \( p!m \), meaning that message \( m \) occurs at in- or output port \( p \). The \( t \)-th step of a trace \( r \) is written \( r_t \); we speak of the step at time \( t \). To capture the aforementioned security property we would require that each output \( (\text{transfer}, d, p, u, v) \) at a port \( \text{PS}_{out}u \) for an honest client \( u \), an honest acquirer \( ac \), and an arbitrary (potentially malicious) merchant \( v \) is preceded by an input \( (\text{pay}, d, p, v) \) at \( \text{PS}_{in}u \). This statement can be significantly strengthened for our payment system by requiring that whenever an arbitrary honest party successfully terminates the protocol execution, the inputs of all honest parties have previously been received. This strengthened variant is called weak atomicity. To simplify notation, let \( \text{SuccessHonestTerm}(d, p, u, v, ac, r, t) \) denote the predicate indicating whether an honest party \( u, v, \) or \( ac \) has successfully terminated the payment protocol for \( d \) and \( p \) in trace \( r \) at time \( t \), i.e., the predicate is defined as the disjunction of \((u \in H \land \text{PS}_{out}u!(\text{paid}, d, p, v) \in r_t), (v \in H \land \text{PS}_{out}v!(\text{received}, d, p, u) \in r_t)\), and \((ac \in H \land \text{PS}_{out}ac!(\text{transfer}, d, p, u, v) \in r_t)\).

**Definition 4.1 (Weak Atomicity)** A trace \( r \) is contained in \( \text{Req}^{\text{weak, atom}} \) if and only if for all \( d \in \Sigma^* \),
\( p \in \mathbb{N}, u \in \mathcal{M}_{\text{client}}, v \in \mathcal{M}_{\text{merchant}}, \text{and} \; t_2 \in \mathbb{N}: \)

\[
\text{SuccessHonestTerm}(d, p, u, v, ac, r, t_2) \Rightarrow \\
\exists t_1, t'_1, t''_1 < t_2: \\
\left( (u \in \mathcal{H} \Rightarrow \text{PS} \text{\_in}_{u}(\text{pay}, d, p, v) \in r_{t_1}) \right) \\
\wedge (v \in \mathcal{H} \Rightarrow \text{PS} \text{\_in}_{v}(\text{receive}, d, p, u) \in r_{t'_1}) \\
\wedge (ac \in \mathcal{H} \Rightarrow \text{PS} \text{\_in}_{ac}(\text{allow}, d, p, u, v) \in r_{t''_1}) \right).}
\]

\( \blacklozenge \)

The main complementary feature of the payment system is its full disputability, i.e., every participant is able to prove that a completed payment has taken place. One can identify two main properties for disputes to be meaningful. First, a party following the protocol wants to be sure that if she initiates a dispute after successfully completing the protocol, the result of the trusted third party has to be true independent of the behavior of other parties. Since the underlying reactive setting grants the adversary full control over the network and in particular to suppress arbitrary messages, we cannot prove statements that “something good” occurs in the future, e.g., that a dispute will be won. We instead formulate the property in a way that allows for backward reasoning. We only formalize the dispute properties for clients; the analogue for merchants and the acquirer can be obtained by simple textual replacement.

**Definition 4.2 (Correct Disputing (Client Part))** A trace \( r \) is contained in \( \text{Req}^\text{corr}\_\text{disp}_{\text{client}} \) if and only if for all \( d \in \Sigma^*, \; p \in \mathbb{N}, \; u \in \mathcal{H}_{\text{client}}, \; v \in \mathcal{M}_{\text{merchant}}, \; \text{and} \; t_3 \in \mathbb{N}: \)

\[
\text{PS} \text{\_out}_{\text{ttp}}!(\text{dispute, paid, false}, d, p, u, v) \in r_{t_3} \\
\Rightarrow \left( \exists t_2 < t_3: (\text{PS} \text{\_in}_{u}(\text{dispute}, d, p, v) \in r_{t_2}) \right) \\
\wedge \forall t_1 < t_2: (\text{PS} \text{\_out}_{u}!(\text{paid}, d, p, v) \notin r_{t_1}) \right).}
\]

\( \blacklozenge \)

Secondly, an honest party wants to be sure that she cannot be blamed for having participated in a payment which she was not involved in, i.e., a dispute for this payment may only be successful if she previously made the corresponding input.

**Definition 4.3 (No Framing (Client Part))** A trace \( r \) is contained in \( \text{Req}^\text{no\_frame}_{\text{client}} \) if and only if for all \( d \in \Sigma^*, \; p \in \mathbb{N}, \; u \in \mathcal{H}_{\text{client}}, \; v \in \mathcal{M}_{\text{merchant}}, \; t_2 \in \mathbb{N}, \; \text{and} \; x \in \{\text{received, transfer}\}: \)

\[
\text{PS} \text{\_out}_{\text{ttp}}?(\text{dispute, x, true}, d, p, u, v) \in r_{t_2} \Rightarrow \\
\exists t_1 < t_2: (\text{PS} \text{\_in}_{u}(\text{pay}, d, p, v) \in r_{t_1}) \right).}
\]

\( \blacklozenge \)

Let \( \text{Req}^\text{corr}\_\text{disp} \) denote the conjunction of \( \text{Req}^\text{corr}\_\text{disp}_{\text{client}} \) and the corresponding properties for merchants and the acquirer; similarly, let \( \text{Req}^\text{no\_frame} \) denote the conjunction of \( \text{Req}^\text{no\_frame}_{\text{client}} \) and its counterparts for merchants and the acquirer.

The notion of a system \( \text{Sys} \) fulfilling an integrity property \( \text{Req} \) essentially comes in two flavors [10]. Perfect fulfillment, \( \text{Sys} \models^{\text{perf}} \text{Req} \), means that the integrity property holds for all traces arising in runs of \( \text{Sys} \) (a well-defined notion from the underlying model [76]). Computational fulfillment, \( \text{Sys} \models^{\text{poly}} \text{Req} \), means that the property only holds for polynomially bounded users and adversaries, and that a negligible error probability is permitted. Perfect fulfillment implies computational fulfillment. The following theorem summarizes what we prove for these requirements:
Theorem 4.1 (Security of the Payment System) Let \( \text{Req}^\text{PS} \):= \( \text{Req}^\text{weak}_\text{atom} \cap \text{Req}^\text{corr}_\text{disp} \cap \text{Req}^\text{no}_\text{frame} \). For the payment system from Section 3.3 we have \( \text{Sys}^\text{PS, id} \models \text{perf} \text{Req}^\text{PS} \) and \( \text{Sys}^\text{PS, real} \models \text{poly} \text{Req}^\text{PS} \). ✷

Note that we did not consider properties concerning the confidentiality of the data involved in the payment. This is similar to the iKP payment system, which does not provide confidentiality itself but instead assumes external mechanisms like underlying secure channels for this task. We can model these mechanisms in our underlying framework as well by inserting secure channel machines between the links of the protocol machines and the cryptographic library, and the corresponding confidentiality properties can then be easily shown.

5 Proof of the Cryptographic Realization from the Idealization

As discussed in the introduction, the idea of our approach is to prove Theorem 4.1 for the payment protocol using the ideal Dolev-Yao-style library. Then the result for the real system follows automatically.

The notion that a system \( \text{Sys}_1 \) securely implements another system \( \text{Sys}_2 \) is called reactive simulatability (recall the introduction), and is written \( \text{Sys}_1 \geq\text{poly}_\text{sec} \text{Sys}_2 \) (in the computational case). The main result of [22] is therefore

\[
\text{Sys}^\text{cry, real} \geq\text{poly}_\text{sec} \text{Sys}^\text{cry, id}.
\]

Since \( \text{Sys}^\text{PS, real} \) and \( \text{Sys}^\text{PS, id} \) are compositions of the same protocol with \( \text{Sys}^\text{cry, real} \) and \( \text{Sys}^\text{cry, id} \), respectively, the composition theorem of [76] and (1) imply

\[
\text{Sys}^\text{PS, real} \geq\text{poly}_\text{sec} \text{Sys}^\text{PS, id}.
\]

(2)

Showing the theorem’s preconditions is easy since the machines \( M^\text{PS}_u \) are polynomial-time (see Section 3.4). Finally, the integrity preservation theorem from [10] and (2) imply for every integrity property \( \text{Req} \)

\[
(\text{Sys}^\text{PS, id} \models \text{poly} \text{Req}) \Rightarrow (\text{Sys}^\text{PS, real} \models \text{poly} \text{Req}).
\]

(3)

Hence if we prove \( \text{Sys}^\text{PS, id} \models \text{perf} \text{Req}^\text{PS} \), we immediately obtain \( \text{Sys}^\text{PS, real} \models \text{poly} \text{Req}^\text{PS} \) from (3).

6 Proof in the Ideal Setting

This section sketches the proof of the ideal part of Theorem 4.1: We prove that the payment protocol implemented with the ideal, Dolev-Yao-style cryptographic library perfectly fulfills the property \( \text{Req}^\text{PS} \). The main challenge in this proof was to find suitable invariants on the state of the ideal payment system.

We start this section with a rigorous definition of the possible states of the ideal cryptographic library as needed for formulating the invariants. We then define the invariants and show how to exploit them to prove the overall integrity property of the payment system. The proof of the overall integrity property is postponed to Appendix A; the proofs of the invariants are omitted due to lack of space.

6.1 Overview and States of the Ideal Cryptographic Library

The ideal cryptographic library administrates Dolev-Yao-style terms and allows each user to operate on them via handles, i.e., via local names specific to this user. The handles also contain the information that knowledge sets give in other Dolev-Yao formalizations: The set of terms that a participant \( u \) knows, including \( u = a \) for the adversary, is the set of terms with a handle for \( u \). As we saw in the payment algorithms, the library offers its user (and the adversary) the typical operations on terms to which they
have handles, e.g., signing with a secret key and signature verification with a public key. The terms are typed; for instance, signature verification only succeeds on signatures and projection only on lists. As secure encryption schemes are necessarily probabilistic and so are most signature schemes, and as the library allows the generation of polynomially many nonces and key pairs, multiple instances of terms of almost every structure can occur, e.g., multiple signatures of the same message \( m \) with the same key \( \text{sks} \). There are multiple ways to deal with this in prior Dolev-Yao models, e.g., counting (for nonces) and multisets. The version in [22] corresponds to counting: The terms are globally numbered by an index. Each term is represented by its type (i.e., root node) and its first-level arguments, which can be indices of earlier terms. This enables easy distinction of, e.g., which of many nonces is signed in a larger term. These global indices are never visible at the user interface. The indices and the handles for each participant are generated by one counter each.

A novel aspect of this cryptographic library compared with prior Dolev-Yao models is that terms have an abstract length parameter, indicating the length of the corresponding real message. It is derived from a tuple \( L \) of length functions that denote how the length of a term depends on the length of its subterms. This is necessary because real encryption cannot entirely hide the length of cleartexts. Moreover, \( L \) contains bounds on the accepted message lengths and the number of accepted inputs at each port. All these bounds can be arbitrary, except that they be polynomially bounded in a security parameter \( k \). Formally, the number \( n \) of participants and the tuple \( L \) are parameters of the system \( \text{Sys}^{\text{cry, id}} \), but we omitted them for readability.

Similarly, \( n \) and a tuple \( L' \) should be parameters of our ideal payment system \( \text{Sys}^{\text{PS, id}} \), see Section 3.4. As the machines \( M^{\text{PS}}_u \) of this system only make bounded-length inputs to the cryptographic library given \( n \) and \( L' \), the bounds in \( L \) can easily be chosen large enough so that all these inputs are legal. Further, as we only prove an integrity properties, it is not a problem in the proof that the number of accepted inputs might be exceeded. This is why we can omit the details of the length functions.

As described above, the terms in the ideal cryptographic library, i.e., in the trusted host \( \text{TH}^{\text{cry, id}}_H \) for every set \( H \) of honest participants, are represented by their top level, and knowledge of them by potential handles for the different participants. The data structure chosen for this in [22] is a database \( D \) in which each entry \( x \) in \( D \) can have the arguments

\[
(x.\text{ind}, x.\text{type}, x.\text{arg}, x.\text{hnd}_{u_1}, \ldots, x.\text{hnd}_{u_m}, x.\text{len}),
\]

where \( H = \{u_1, \ldots, u_m\} \) and the arguments have the following types and meaning:

- \( x.\text{ind} \) is the global index of an entry.
- \( x.\text{type} \in \text{typeset} \) identifies the type of \( x \). The types nonce, list, data (for payload data), sks and pks (for secret and public signature keys), and sig (for signatures) occur in the following.
- \( x.\text{arg} = (a_1, a_2, \ldots, a_j) \) is a possibly empty list of arguments. Arguments of type \( \text{INDS} \) are indices of other entries (subterms); we sometimes distinguish them by a superscript “\( \text{ind} \)”.
- \( x.\text{hnd}_u \in \text{HND} \cup \{\downarrow\} \) for \( u \in H \cup \{a\} \) are handles, where \( x.\text{hnd}_u = \downarrow \) means that \( u \) does not know this entry.
- \( x.\text{len} \in \mathbb{N}_0 \) denotes the length of the entry.

The machine \( \text{TH}^{\text{cry}}_H \) has a counter \( \text{size} \in \text{INDS} \) for the current size of \( D \) and counters \( \text{curhnd}_u \) (current handle) for the handles, all initialized with 0.
The assumption that keys have already been generated and distributed means that for each \( u \in \mathcal{M} \) two entries of the following form are added to \( D \), where \( \{u_1, \ldots, u_m\} := \mathcal{H} \):

\[
(sks_u, \text{type} := \text{sks}, \text{arg} := (0), \text{hnd}_u := \text{sks}^\text{hnd}_u, \\
\text{len} := 0);
\]

\[
(pks_u, \text{type} := \text{pks}, \text{arg} := (), \\
\text{hnd}_{u_1} := \text{pks}^\text{hnd}_{u_1, u}, \ldots, \text{hnd}_{u_m} := \text{pks}^\text{hnd}_{u_m, u}, \\
\text{hnd}_u := \text{pks}^\text{hnd}_u, \text{len} := \text{pks}^\text{len}(k)).
\]

Here \( sks_u \) and \( pks_u \) are consecutive natural numbers, \( \text{pks}^\text{len}(k) \) is the length function for public keys, and the argument of the secret key entry stores the number of messages that have already been signed with this key. Treating secret keys as being of length 0 is a technicality in [22] and will not matter in the sequel.

### 6.2 Invariants

This section contains invariants of the system \( \text{Sys}^{\text{PS}, \text{id}} \), which are used in the proof of the ideal part of Theorem 4.1. The first invariant, no modification, states that attributes of database entries of honest users are never changed after they have been initialized. This is true for all attributes except for status, which may additionally change to processed. We let \( r_t : D^\text{PS}_u \) denote the contents of database \( D^\text{PS}_u \) at time \( t \) in trace \( r \).

**Invariant 1 (No Modification)** For all \( u \in \mathcal{H} \), \( i \in \mathcal{INS} \), \( t_1 \in \mathbb{N} \), all traces \( r \) arising in runs of \( \text{Sys}^{\text{PS}, \text{id}} \), and \( s := t_1 : D^\text{PS}_u[i].\text{status} \):

\[
(r_{t_1} : D^\text{PS}_u[i].x \neq \downarrow \land x \neq \text{status} \Rightarrow \\
\forall t_2 > t_1 : (r_{t_2} : D^\text{PS}_u[i].x = r_{t_1} : D^\text{PS}_u[i].x)) \\
\land \ (s \in \{\text{pay}, \text{invoice}, \text{allow}, \text{auth_request}\} \Rightarrow \\
\forall t_2 > t_1 : (r_{t_2} : D^\text{PS}_u[i].\text{status} \in \{s, \text{processed}\})).
\]

The next invariant, unique payment entries, establishes that entries in the databases of honest parties are uniquely determined by the price, the description, the client, and the merchant. We state this and the following invariants exemplarily for honest clients.

**Invariant 2 (Unique Payment Entries (Client Part))** For all \( u \in \mathcal{H} \), \( i_1, i_2 \in \mathcal{INS} \), \( t \in \mathbb{N} \), and all traces \( r \) arising in runs of \( \text{Sys}^{\text{PS}, \text{id}} \):

\[
(r_t : D^\text{PS}_u[i_1].\text{price} = r_t : D^\text{PS}_u[i_2].\text{price} \neq \downarrow \\
\land r_t : D^\text{PS}_u[i_1].\text{desc} = D^\text{PS}_u[i_2].\text{desc} \neq \downarrow \\
\land r_t : D^\text{PS}_u[i_1].\text{merch} = D^\text{PS}_u[i_2].\text{merch} \neq \downarrow \Rightarrow \\
(i_1 = i_2).
\]

The next invariant, correct signing, characterizes which lists will be signed by an honest party. It states that all signed lists in the database \( D \) of the ideal cryptographic library that start with a payment element are of the form \( (\text{payment}, d, p, u, v) \), and that a local command \( (\text{pay}, d, p, v) \) must have been received before the entry was created. We use explanatory comments in the definition of the invariant to increase readability.
$r_{t_2} : D[i].type = \text{sig} \land r_{t_2} : D[i].arg[1] = pks_u \land$  

$\exists t_1 < t_2 : (\text{PS}_u?((\text{pay}, x_2, x_3, x_5) \in r_{t_1}))$.  

# If a payment message is signed with u's key, then # the message is of the correct format and # a matching input has occurred.

\begin{figure}
\begin{align*}
\text{PS}_u!(\text{paid}, d, p, v) & \in r_t \Rightarrow \\
\left( r_t : D[h\text{nd}_u = s^m].type = \text{sig} \land \\
r_t : D[pk^m].h\text{nd}_u = pks^m_{u,v} \land \\
r_t : D[h\text{nd}_u = s^ac].type = \text{sig} \land \\
r_t : D[pk^ac].h\text{nd}_u = pks^ac_{u,ac} \land \\
x^m_1 = \text{invoice} \land x^m_2 = d \land x^m_3 = p \land x^m_5 = v \land \\
x^{ac}_1 = \text{auth}\_\text{response} \land x^{ac}_2 = d \land x^{ac}_3 = p \land x^{ac}_5 = v \right).
\end{align*}
\end{figure}

# If u terminates the protocol, # the signatures of # the merchant and # the signature of # the acquirer are stored, # and they range # over the correct data.

\begin{figure}
\begin{align*}
\text{Invariant 3} & \text{ (Correct Signing (Client Part)) For all } u \in \mathcal{M}^\text{client} \cap \mathcal{H}, i \in \mathcal{I}D\mathcal{S}, t_2 \in \mathbb{N}, \text{ and all traces } r \text{ arising in runs of } \text{Sys}^{PS,\text{id}}, \text{ the formula in Figure 3 holds, where } l := r_{t_2} : D[i].\text{arg}[2] \text{ denotes the index of the signed list, } y_j := r_{t_2} : D[l].\text{arg}[j] \text{ the indices of the list elements, and } x_j := r_{t_2} : D[y_j].\text{arg}[1] \text{ the actual data for } j = 1, \ldots, 5. \\
\end{align*}
\end{figure}

In the proof of the overall integrity property, we will later show that certain outputs of honest users may only occur after signatures of specific messages created with specific keys have been received, e.g., that the machine of an honest acquirer only outputs an allow message if it formerly received a signature that is valid with respect to the respective client’s public key. Then correct signing will allow us to deduce that the message is of the correct format and that a previous input by the client must have occurred.

The last invariant, correct storing, captures that an honest user has stored the signatures of both remaining parties if it successfully terminates the protocol. We again show the invariant exemplarily for honest clients; it states that if the machine of a client u outputs (paid, d, p, v) then it correctly stored the signatures of both remaining parties, i.e., $\text{sig}_m$ is a signature of a list (invoice, d, p, u, v) signed by v and $\text{sig}_ac$ is a signature of a list (auth\_response, d, p, u, v) signed by ac.

\begin{figure}
\begin{align*}
\text{Invariant 4} & \text{ (Correct Storing (Client Part)) For all } u \in \mathcal{M}^\text{client} \cap \mathcal{H}, d \in \Sigma^*, p \in \mathbb{N}, v \in \mathcal{M}^\text{merchant}, \\
t \in \mathbb{N}, \text{ and all traces } r \text{ arising in runs of } \text{Sys}^{PS,\text{id}}, \text{ the formula in Figure 4 holds, where for } w \in \{m, ac\}, \text{ we let } s^w := r_t : D^u[s^w\{\text{desc} = d \land \text{price} = p \land \text{merchant} = v\}.\text{sig}_w \text{ denote the handle to w’s signature, } \\
pk^w := r_t : D[\text{hnd}_u = s^w].\text{arg}[1] \text{ the index of the public key used, } l^w := r_t : D[\text{hnd}_u = s^w].\text{arg}[2] \text{ the index of the signed list, } y^w_j := r_t : D[l^w].\text{arg}[j] \text{ the indices of the list elements, and } x^w_j := r_t : D[y^w_j].\text{arg}[1] \text{ the actual data for } j = 1, \ldots, 5.
\end{align*}
\end{figure}
This invariant is key for proving the disputability properties of the protocol since it implies that dispute messages sent by honest users are always of a specific format and the contained signatures are valid with respect to specific public keys. Based on this, we can easily infer the output of the trusted third party.

7 Conclusion and Outlook

We have proven an electronic payment system to be secure in the real cryptographic setting. The payment system is a slightly simplified variant of the 3KP payment system and comprises a variety of different security requirements ranging from the impossibility of unauthorized payments and weak atomicity to more sophisticated properties like disputability. The proof was done by exploiting a Dolev-Yao-style deterministic idealization of cryptography which has a provably secure real cryptographic implementation in the sense that security is reduced to the hardness of underlying computational assumptions. Composition and integrity preservation theorems from the underlying model imply that the protocol proof with the idealized cryptography carries over to the real protocol implementation. This was the first example of such a proof for protocols involving digital signatures, and for Dolev-Yao style payment systems in general. In spite of certain differences to usual Dolev-Yao variants, in particular a representation of terms or real cryptographic objects to the protocol layer by handles (local names) and length functions in the idealization, the proof seems to be of a type readily accessible to automatic proof tools.

References


A Postponed Proofs
Algorithm 4 Merchant: Evaluation of User Inputs for Receiving in $M^\text{PS}_u$

**Input:** (receive, d, p, v) at $\text{PS}_{\text{in}} u$? with $d \in \Sigma^*$, $p \in \mathbb{N}$, and $v \in \mathcal{M}_{\text{client}}$.

1. if $D^\text{PS}_u[\text{desc} = d \land \text{price} = p \land \text{client} = v] = \downarrow$ then
2. $D^\text{PS}_u := (\text{curr}_{\text{in}} d++, d, p, v, \downarrow, \downarrow)$. 
3. invoice$^\text{hnd} \leftarrow \text{store}($invoice$)$.
4. $d^\text{hnd} \leftarrow \text{store}($d$)$.
5. $p^\text{hnd} \leftarrow \text{store}($p$)$.
6. $u^\text{hnd} \leftarrow \text{store}($u$)$.
7. $v^\text{hnd} \leftarrow \text{store}($v$)$.
8. $l^\text{hnd} \leftarrow \text{list}(\text{invoice}^\text{hnd}, d^\text{hnd}, p^\text{hnd}, v^\text{hnd}, a^\text{hnd})$.
9. $s^\text{hnd} \leftarrow \text{sign}(l^\text{hnd})$.
10. $m^\text{hnd} \leftarrow \text{list}(\text{invoice}^\text{hnd}, s^\text{hnd})$.
11. send($u, m^\text{hnd}$).
12. end if

A.1 Proof of the Overall Integrity Property

**Proposition A.1** For the payment system from Section 3.3 and the weak atomicity property $\text{Req}^{\text{weak}_{\text{atom}}}$, we have $\text{Sys}^{\text{PS}_{\text{id}}} \models \text{perf} \text{Req}^{\text{weak}_{\text{atom}}}$.

**Proof:** We only give the proof for the client part of weak atomicity, i.e., we prove the statement $\text{SuccessHonestTerm}(d, p, u, v, \text{ac}, r, t_2) \land u \in \mathcal{H} \Rightarrow \exists t_1 < t_2: \text{PS}_{\text{in} u}?(\text{pay}, d, p, v) \in r_{t_1}$. The other parts can be proved similarly. Let $d \in \Sigma^*$, $p \in \mathbb{N}$, $u \in \mathcal{M}_{\text{client}} \cap \mathcal{H}$, $v \in \mathcal{M}_{\text{merchant}}$, $t_2 \in \mathbb{N}$, and $r$ a trace arising in runs of $\text{Sys}^{\text{PS}_{\text{id}}}$, and $i := r_{t_2} : D^\text{PS}_u[\text{desc} = d \land \text{price} = p \land \text{merchant} = v].\text{ind}$. Recall that $\text{SuccessHonestTerm}(d, p, u, v, \text{ac}, r, t_2)$ is defined as the disjunction of $(u \in \mathcal{H} \land \text{PS}_{\text{out} v}!(\text{paid}, d, p, v) \in r_{t_2})$, $(v \in \mathcal{H} \land \text{PS}_{\text{out} v}!(\text{received}, d, p, u) \in r_{t_2})$, and $(\text{ac} \in \mathcal{H} \land \text{PS}_{\text{out} v}!(\text{transfer}, d, p, u, v) \in r_{t_2})$. We will prove these three cases separately.

Assume $\text{PS}_{\text{out} v}!(\text{paid}, d, p, v) \in r_{t_2}$. This output may only occur in Step 2.28. Step 2.25 and 2.26 together with unique payment entries and no modification implies $r_{t_2} : D^\text{PS}_u[i].\text{status} = \text{processed}$, and the status of $D^\text{PS}_u[i]$ may only have been set to processed in Step 1.3 or 2.11. In the first case, Step 1.1 immediately implies that there the algorithm was activated on input $(\text{pay}, d, p, v)$ at port $\text{PS}_{\text{in} u}?$ at some time $t_1 < t_2$. In the second case, Step 2.9 ensures $r_{t_2} : D^\text{PS}_u[i].\text{status} = \text{pay}$ for some $t_1' < t_2$, hence this entry was created in Step 1.14. Again the algorithm was activated on inputs $(\text{pay}, d, p, v)$ at port $\text{PS}_{\text{in} u}?$ at some time $t_1 < t_2 < t_2'$. Assume $\text{PS}_{\text{out} v}!(\text{received}, d, p, u) \in r_{t_2}$ and $v \in \mathcal{H}$. This output may only occur in Step 6.28. Let $j := r_{t_2} : D^\text{PS}_u[\text{desc} = d \land \text{price} = p \land \text{client} = u].\text{ind}$. Step 6.23 and 6.24 together with unique payment entries and no modification imply $r_{t_2} : D^\text{PS}_u[j].\text{sig}_c \neq \downarrow$. The only step where $v$ assigns a value different from $\downarrow$ to attribute $\text{sig}_c$ is in Step 6.10. Let $l^\text{hnd} = r_{t_2} : D^\text{PS}_u[j].\text{sig}_c$ and let $i := r_{t_2} : D[\text{hnd}_v = l^\text{hnd}].\text{ind}$. Let $l := r_{t_2} : D[i].\text{arg}[2], y_j := r_{t_2} : D[i].\text{arg}[j]$, and $x_j := r_{t_2} : D[y_j].\text{arg}[1]$ for $j = 1, \ldots, 5$. Then Steps 6.4-6.5, the definition of the commands verify, list$._\text{proj}$, and retrieve imply $r_{t_2} : D[i].\text{type} = \text{sig}$, $r_{t_2} : D[i].\text{arg}[1] = pks_u$, $x_1 = \text{payment}$, $x_2 = d$, $x_3 = p$, and $x_5 = v$. Hence the entry $D[i]$ fulfills the requirements of the correct signing, thus there exists $t_1 < t_2$ such that $\text{PS}_{\text{in} u}?(\text{pay}, d, p, v) \in r_{t_1}$.

Assume $\text{PS}_{\text{out} \text{ac}!}(\text{transfer}, d, p, u, v) \in r_{t_2}$ and $\text{ac} \in \mathcal{H}$. This output may only occur in Step 8.23. This case can be proven exactly as the previous one with the corresponding steps of Algorithm 8.

**Proposition A.2** For the payment system from Section 3.3 and the correct disputing property $\text{Req}^{\text{corr}_{\text{disp}}}$, we have $\text{Sys}^{\text{PS}_{\text{id}}} \models \text{perf} \text{Req}^{\text{corr}_{\text{disp}}}$.
Algorithm 5 Acquirer: Evaluation of User Inputs for Allow in $M_{ac}^{PS}$

Input: (allow, $d$, $p$, $u$, $v$) at $PS_{\text{in}_{ac}}$? with $d \in \text{CHARSET}$, $p \in \mathbb{N}$, $u \in M^{\text{client}}$, and $v \in M^{\text{merchant}}$.

1: $i := D_{ac}^{PS}([\text{desc} = d \land \text{price} = p \land \text{client} = u \land \text{merch} = v])$.ind.
2: if $i \neq \downarrow \land D_{ac}^{PS}[i].\text{status} = \text{auth\_request}$ then
3: $D_{ac}^{PS}[i].\text{status} := \text{processed}.$
4: $\text{auth\_response}^{\text{hnd}} \leftarrow \text{store(\text{auth\_response})}. \quad \text{\(D_{ac}^{PS}[i].\text{status} := \text{processed}.$\)
5: $d^{\text{hnd}} \leftarrow \text{store}(d). \quad \text{\(D_{ac}^{PS}[i].\text{status} := \text{processed}.$\)
6: $p^{\text{hnd}} \leftarrow \text{store}(p). \quad \text{\(D_{ac}^{PS}[i].\text{status} := \text{processed}.$\)
7: $u^{\text{hnd}} \leftarrow \text{store}(u). \quad \text{\(D_{ac}^{PS}[i].\text{status} := \text{processed}.$\)
8: $v^{\text{hnd}} \leftarrow \text{store}(v). \quad \text{\(D_{ac}^{PS}[i].\text{status} := \text{processed}.$\)
9: $l^{\text{hnd}} \leftarrow \text{list(\text{auth\_response}^{\text{hnd}}, d^{\text{hnd}}, p^{\text{hnd}}, u^{\text{hnd}}, v^{\text{hnd}})}.$
10: $\text{sign}^{\text{hnd}}(\text{sks}_{\text{ac}}, l^{\text{hnd}}). \quad \text{\(D_{ac}^{PS}[i].\text{status} := \text{processed}.$\)
11: $m^{\text{hnd}} \leftarrow \text{list(\text{auth\_response}^{\text{hnd}}, a^{\text{hnd}})}.$
12: $\text{send}_{i}(v, m^{\text{hnd}}). \quad \text{\(D_{ac}^{PS}[i].\text{status} := \text{processed}.$\)
13: else if $i = \downarrow$ then
14: $D_{ac}^{PS} := (\text{curr\_ind}_{ac}++, d, p, u, v, \downarrow, \downarrow, \text{allow}). \quad \text{\(D_{ac}^{PS}[i].\text{status} := \text{processed}.$\)
15: end if

Proof. Let $d \in \Sigma^{*}$, $p \in \mathbb{N}$, $u \in M^{\text{client}} \cap H$, $v \in M^{\text{merchant}}$, $t_{3} \in \mathbb{N}$, and $r$ a trace arising in runs of $Sys_{\text{PS}}^{\text{id}}$. Again, we only show the client part of the statement, i.e., assume $PS_{\text{out}_{tp}}(\text{dispute, paid, false, } d, p, u, v) \in r_{t_{3}}$. The output must have occurred in Step 10.20.

The algorithm is invoked only on input $(w, ttp, i, l^{\text{hnd}})$. Let $i := D[\text{hnd}_{tp} = l^{\text{hnd}}].\text{arg}[1]$, $l_{1} := D[i].\text{arg}[2]$, $(pk, s_{1}, s_{2}) := D[l_{1}].\text{arg}[1, 2, 3]$, $y_{j} := D[s_{1}].\text{arg}[j]$, and $x_{j} := D[y_{j}].\text{arg}[1]$ for $j = 1, \ldots, 5$. Step 10.17 implies $w = x_{4} = u$.

Now Step 10.1-8 and the definition of the commands verify, list\_proj, and msg\_of\_sig imply $D[i].\text{type} = \text{sig}$ and $pk = pks_{u}$. It can then be shown along the lines of the proof of correct signing that $M_{ac}^{PS}$ must have input a command $\text{sign}(\text{dispute, } D[s_{1}], hnd_{u}, D[s_{2}], hnd_{v})$ such that $x_{1} = \text{payment}$, $x_{2} = d$, $x_{3} = p$, $x_{4} = u$, and $x_{5} = v$. The only syntactically matching sign command is in Step 3.4, and is executed only on input $(\text{dispute, } d, p, v)$ at $PS_{\text{in}_{ac}}$? at some time $t_{2} < t_{3}$. We remain to show the nonexistence of an output $(\text{paid, } d, p, v)$ at $PS_{\text{out}_{u}}$ for all times $t_{1} < t_{2}$. We prove this by contradiction. Assume that there exists $t_{1} \in \mathbb{N}$ such that $PS_{\text{out}_{u}}(\text{paid, } d, p, v) \in r_{t_{1}}$. Then by correct storing both signatures $\text{sig}_{m}$ and $\text{sig}_{ac}$ have been stored and the condition in Step 10.17 can easily shown to be true, i.e., the output of Step 10.20 will never occur. (This last step can be made more formal but requires re-stating the whole formalism of the invariant and seems to complicate understanding here.)

Proposition A.3 For the payment system from Section 3.3 and the no framing property $\text{Req}_{\text{no\_frame}}$, we have $Sys_{\text{PS}}^{\text{id}} \models \text{perf}$ $\text{Req}_{\text{no\_frame}}$.

Proof. Let $x \in \{\text{received, transfer}\}$, $d \in \Sigma^{*}$, $p \in \mathbb{N}$, $u \in H^{\text{client}}$, $v \in M^{\text{merchant}}$, $t_{2} \in \mathbb{N}$, and $r$ a trace arising in runs of $Sys_{\text{PS}}^{\text{id}}$. Again we prove only the client part of the statement, i.e., we prove the statement $PS_{\text{out}_{tp}}(\text{dispute, } d, p, u, v) \in r_{t_{2}} \Rightarrow \exists t_{1} < t_{2} : PS_{\text{in}_{ac}}?(\text{pay, } d, p, v) \in r_{t_{1}}$.

Assume $PS_{\text{out}_{tp}}(\text{dispute, } x, \text{true, } d, p, u, v) \in r_{t_{3}}$ for $x = \text{received (} x = \text{transferr).}$ This output occurs only in Step 10.26 (in Step 10.34). With $l_{2}^{\text{hnd}}$ as in Algorithm 10, let $i := r_{t_{2}} : D[\text{hnd}_{tp} = l_{2}^{\text{hnd}}].\text{ind}$. Now Step 10.23 and 10.25 (Step 10.31 and 10.33) ensure that $D[i].\text{type} = \text{sig}$, $D[i].\text{arg}[1] = pks_{x_{4}} = D[i].\text{arg}[1] = pks_{u}$, and $x_{1} = \text{payment}$. This implies that $D[i]$ meets the prerequisites correct signing, hence there exists $t_{1} < t_{2}$ such that $PS_{\text{in}_{ac}}?(\text{pay, } d, p, v) \in r_{t_{1}}$.
Algorithm 6 Merchant: Evaluation of Inputs from the Cryptographic Library in $M_{\text{u}}^{\text{PS}}$

Input: $(v, u, i, l_{\text{hnd}})$ at out$_u$? with $v \in \mathcal{M}_{\text{client}} \cup \{\text{ac}\}$.

1: $l_{i_{\text{hnd}}} \leftarrow \text{list}_\text{proj}(l_{\text{hnd}}, j)$ for $j = 1, 2$.
2: $l_1 \leftarrow \text{retrieve}(l_{i_{\text{hnd}}})$.
3: if $l_1 = \text{payment} \land v \neq \text{ac}$ then
4: \hspace{1em} $m_{2_{\text{hnd}}} \leftarrow \text{msg}_\text{of}_\text{sig}(l_{2_{\text{hnd}}})$.
5: \hspace{1em} $b \leftarrow \text{verify}(l_{2_{\text{hnd}}}, \text{pk}_{u_{\text{hnd}}}, m_{2_{\text{hnd}}})$.
6: \hspace{1em} $x_j \leftarrow \text{list}_\text{proj}(m_{2_{\text{hnd}}}, j)$ for $j = 1, \ldots, 5$.
7: \hspace{1em} $x_j \leftarrow \text{retrieve}(x_j)$ for $j = 1, \ldots, 5$.
8: \hspace{1em} $i := D_{\text{u}}^{\text{PS}}[\text{desc} = x_2 \land \text{price} = x_3 \land \text{client} = x_4 \land \text{sig}_c = \downarrow].\text{ind}$.
9: \hspace{1em} if $x_1 = \text{payment} \land x_4 = v \land x_5 = u \land b = \text{true} \land i \neq \downarrow$ then
10: \hspace{2em} $D_{\text{u}}^{\text{PS}}[i].\text{sig}_c := l_{2_{\text{hnd}}}$.
11: \hspace{2em} if $x_1 = \text{auth}_\text{response} \land v = \text{ac}$ then
12: \hspace{3em} $m_{2_{\text{hnd}}} \leftarrow \text{msg}_\text{of}_\text{sig}(l_{2_{\text{hnd}}})$.
13: \hspace{3em} $b \leftarrow \text{verify}(l_{2_{\text{hnd}}}, \text{pk}_{u_{\text{hnd}}}, m_{2_{\text{hnd}}})$.
14: \hspace{3em} $x_j \leftarrow \text{list}_\text{proj}(m_{2_{\text{hnd}}}, j)$ for $j = 1, \ldots, 5$.
15: \hspace{3em} $x_j \leftarrow \text{retrieve}(x_j)$ for $j = 1, \ldots, 5$.
16: \hspace{2em} end if
17: \hspace{1em} else if $x_1 = \text{auth}_\text{response} \land v = \text{ac}$ then
18: \hspace{2em} \hspace{1em} $i := D_{\text{u}}^{\text{PS}}[\text{desc} = x_2 \land \text{price} = x_3 \land \text{client} = x_4 \land \text{sig}_c \neq \downarrow].\text{ind}$.
19: \hspace{2em} \hspace{1em} if $x_5 = u \land b = \text{true} \land i \neq \downarrow$ then
20: \hspace{3em} \hspace{2em} $D_{\text{u}}^{\text{PS}}[i].\text{sig}_c \leftarrow l_{2_{\text{hnd}}}$.
21: \hspace{3em} \hspace{2em} confirm$_{\text{hnd}} \leftarrow \text{store}(\text{confirm})$.
22: \hspace{3em} \hspace{2em} $m_{\text{hnd}} \leftarrow \text{list}(\text{confirm}_\text{hnd}, i_{\text{hnd}}, s_{\text{hnd}})$.
23: \hspace{2em} \hspace{2em} Output (received, $x_2, x_3, x_4$) at PS$_\text{out}_u!$.
24: \hspace{2em} \hspace{1em} end if
25: \hspace{1em} end if
26: end if

Algorithm 7 Merchant: Evaluation of User Inputs for Disputes in $M_{\text{u}}^{\text{PS}}$

Input: (dispute, $d, p, v$) at PS$_\text{in}_u$? with $d \in \Sigma^*, p \in \mathbb{N}$, and $v \in \mathcal{M}_{\text{client}}$.

1: if $i := D_{\text{u}}^{\text{PS}}[\text{desc} = d \land \text{price} = p \land \text{client} = v \land \text{sig}_c \neq \downarrow \land \text{sig}_\text{ac} \neq \downarrow].\text{ind} \neq \downarrow$ then
2: \hspace{1em} dispute$_{\text{hnd}} \leftarrow \text{store}(\text{dispute})$.
3: \hspace{1em} $l_{i_{\text{hnd}}} \leftarrow \text{list}(\text{dispute}_{\text{hnd}}, D_{\text{u}}^{\text{PS}}[i].\text{sig}_c, D_{\text{u}}^{\text{PS}}[i].\text{sig}_\text{ac})$.
4: \hspace{1em} $s_{\text{hnd}} \leftarrow \text{sign}(\text{sk}_{u_{\text{hnd}}}, l_{\text{hnd}})$.
5: \hspace{1em} $m_{\text{hnd}} \leftarrow \text{list}(s_{\text{hnd}})$.
6: \hspace{1em} send$_i$(ttp, $m_{\text{hnd}}$).
7: end if
Algorithm 8 Acquirer: Evaluation of User Inputs from the Cryptographic Library in \( \text{MP}^{\text{PS}}_{\text{ac}} \)

**Input:** \((v, ac, i, l_{\text{hand}})\) at \(\text{out}_{\text{ac}}\)? with \(v \in \mathcal{M}_{\text{merchant}}\).

1. \(l_{\text{hand}}^j \leftarrow \text{list_proj}(l_{\text{hand}}, j)\) for \(j = 1, 2, 3\).
2. \(l_1 \leftarrow \text{retrieve}(l_{\text{hand}}^1)\).
3. if \(l_1 \neq \text{auth_request}\) then
   4.   Abort
5. end if
6. \(m_{\text{hand}}^j \leftarrow \text{msg_of_sig}(l_{\text{hand}}^j)\) for \(j = 2, 3\).
7. \(x_j \leftarrow \text{list_proj}(m_{\text{hand}}^j, j)\) for \(j = 1, \ldots, 5\).
8. \(x_j \leftarrow \text{retrieve}(x_j)\) for \(j = 1, \ldots, 5\).
9. \(y_j \leftarrow \text{list_proj}(m_{\text{hand}}^j, j)\) for \(j = 1, \ldots, 5\).
10. \(y_j \leftarrow \text{retrieve}(y_j)\) for \(j = 1, \ldots, 5\).
11. \(b_2 \leftarrow \text{verify}(l_{\text{hand}}^2, pk_{\text{ac}, i_1}, m_{\text{hand}}^2)\).
12. \(b_3 \leftarrow \text{verify}(l_{\text{hand}}^3, pk_{\text{ac}, v}, m_{\text{hand}}^3)\).
13. if \(x_1 = \text{payment} \wedge y_1 = \text{invoice} \wedge b_2 = b_3 = \text{true} \wedge x_5 = v \wedge \forall j = 2, \ldots, 5: x_j = y_j\) then
14.   \(i := D_{\text{PS}}^\text{ac}([\text{desc} = x_2 \wedge \text{price} = x_3 \wedge \text{client} = x_4 \wedge \text{merch} = x_5])\).
15. if \(i \uparrow \text{for } D_{\text{ac}}[i].\text{status} = \text{allow}\) then
16.   \(D_{\text{ac}}[i].\text{sig}_c := l_{\text{hand}}^2\).
17.   \(D_{\text{ac}}[i].\text{sig}_m := l_{\text{hand}}^3\).
18.   \(D_{\text{ac}}[i].\text{status} := \text{processed}\).
19.   \(\text{auth_response}_{\text{hand}} \leftarrow \text{store(\text{auth_response})}\).
20.   \(m_{\text{hand}}^1 \leftarrow \text{list(\text{auth_response}_{\text{hand}}, x_{\text{hand}}^2, x_{\text{hand}}^3, x_{\text{hand}}^4, x_{\text{hand}}^5)}\).
21.   \(s_{\text{hand}} \leftarrow \text{sign} (sk_{\text{ac}}, m_{\text{hand}}^1)\).
22.   \(m_{\text{hand}} \leftarrow \text{list(\text{auth_response}_{\text{hand}}, s_{\text{hand}})}\).
23. Output (\text{transfer}, \(x_2, x_3, x_4, x_5\)) at \(\text{PS}_{\text{out}_{\text{ac}}}\).
24. \(\text{send(i}(v, m_{\text{hand}})\).
25. else if \(i = \downarrow\) then
26.   \(D_{\text{ac}} : = (\text{cur_ind}_{\text{ac}} + 1, x_2, x_3, x_4, x_5, l_{\text{hand}}^1, l_{\text{hand}}^2, \text{auth_request})\).
27. end if
28. end if

Algorithm 9 Acquirer: Evaluation of User Inputs for Disputes in \( \text{MP}^{\text{PS}}_{\text{ac}} \)

**Input:** (dispute, \(d, p, u, v\)) at \(\text{PS}_{\text{in}_{\text{ac}}}\)? with \(d \in \Sigma^*, p \in \mathbb{N}, u \in \mathcal{M}_{\text{client}}\), and \(v \in \mathcal{M}_{\text{merchant}}\).

1. \(i := D_{\text{PS}}^\text{ac}([\text{desc} = d \wedge \text{price} = p \wedge \text{client} = u \wedge \text{merch} = v \wedge \text{sig}_c \neq \downarrow \wedge \text{sig}_m \neq \downarrow ]).\text{ind} \neq \downarrow\) then
2. \(\text{dispute}_{\text{hand}} \leftarrow \text{store(\text{dispute})}\).
3. \(l_{\text{hand}} \leftarrow \text{list(\text{dispute}_{\text{hand}}, D_{\text{ac}}[i].\text{sig}_c, D_{\text{ac}}[i].\text{sig}_m)}\).
4. \(s_{\text{hand}} \leftarrow \text{sign} (sk_{\text{ac}}, l_{\text{hand}})\).
5. \(m_{\text{hand}} \leftarrow \text{list(s_{\text{hand})}}\).
6. \(\text{send(i}(\text{tp}, m_{\text{hand}})\).
7. end if
Algorithm 10 TTP: Evaluation of Inputs from the Cryptographic Library in $M_{\text{tup}}^{\text{PS}}$

**Input:** $(v, \text{ttp}, i, \tilde{\text{hnd}})$ at $\text{out}_{\text{tup}}$ for $v \in M \setminus \{\text{ttp}\}$.

1. $s^{\text{hnd}} \leftarrow \text{list}_\text{proj}(\tilde{\text{hnd}}, 1)$.
2. $t^{\text{hnd}} \leftarrow \text{msg}_\text{of}_\text{sig}(s^{\text{hnd}})$.
3. $b_1 \leftarrow \text{verify}(s^{\text{hnd}}, \text{pk}^{\text{hnd}}_{\text{tup}, \text{v}}, t^{\text{hnd}})$.
4. $l_j^{\text{hnd}} \leftarrow \text{list}_\text{proj}(l_j^{\text{hnd}}, j)$ for $j = 1, 2, 3$.
5. $l_1 \leftarrow \text{retrieve}(l_1^{\text{hnd}})$.
6. **if** $l_1 \neq \text{dispute} \lor b_1 \neq \text{true} \text{ then}$
   7. **Abort**
8. **end if**
9. $m_j^{\text{hnd}} \leftarrow \text{msg}_\text{of}_\text{sig}(l_j^{\text{hnd}})$ for $j = 2, 3$.
10. $x_j^{\text{hnd}} \leftarrow \text{list}_\text{proj}(m_j^{\text{hnd}}, j)$ for $j = 1, \ldots, 5$.
11. $x_j \leftarrow \text{retrieve}(x_j^{\text{hnd}})$ for $j = 1, \ldots, 5$.
12. $y_j^{\text{hnd}} \leftarrow \text{list}_\text{proj}(m_j^{\text{hnd}}, j)$ for $j = 1, \ldots, 5$.
13. $y_j \leftarrow \text{retrieve}(y_j^{\text{hnd}})$ for $j = 1, \ldots, 5$.
14. **if** $v \in M^{\text{client}}$ **then**
   15. $b_2 \leftarrow \text{verify}(l_2^{\text{hnd}}, \text{pk}^{\text{hnd}}_{\text{tup}, \text{c}}, m_2^{\text{hnd}})$.
   16. $b_3 \leftarrow \text{verify}(l_3^{\text{hnd}}, \text{pk}^{\text{hnd}}_{\text{tup}, \text{c}}, m_3^{\text{hnd}})$.
   17. **if** $x_1 = \text{invoice} \land y_1 = \text{auth}_\text{response} \land x_4 = v \land b_2 = b_3 = \text{true} \land \forall j = 2, \ldots, 5: x_j = y_j$ **then**
      18. **Output** (dispute, paid, true, $x_2, x_3, x_4, x_5$) at $\text{PS}_\text{out}_{\text{tup}}$.
   19. **else**
      20. **Output** (dispute, paid, false, $x_2, x_3, x_4, x_5$) at $\text{PS}_\text{out}_{\text{tup}}$.
   21. **end if**
22. **else if** $v \in M^{\text{merchant}}$ **then**
   23. $b_2 \leftarrow \text{verify}(l_2^{\text{hnd}}, \text{pk}^{\text{hnd}}_{\text{tup}, \text{m}}, m_2^{\text{hnd}})$.
   24. $b_3 \leftarrow \text{verify}(l_3^{\text{hnd}}, \text{pk}^{\text{hnd}}_{\text{tup}, \text{m}}, m_3^{\text{hnd}})$.
   25. **if** $x_1 = \text{payment} \land y_1 = \text{auth}_\text{response} \land x_5 = v \land b_2 = b_3 = \text{true} \land \forall j = 2, \ldots, 5: x_j = y_j$ **then**
      26. **Output** (dispute, received, true, $x_2, x_3, x_4, x_5$) at $\text{PS}_\text{out}_{\text{tup}}$.
   27. **else**
      28. **Output** (dispute, received, false, $x_2, x_3, x_4, x_5$) at $\text{PS}_\text{out}_{\text{tup}}$.
   29. **end if**
30. **else if** $v = \text{ac}$ **then**
   31. $b_2 \leftarrow \text{verify}(l_2^{\text{hnd}}, \text{pk}^{\text{hnd}}_{\text{tup}, \text{c}}, m_2^{\text{hnd}})$.
   32. $b_3 \leftarrow \text{verify}(l_3^{\text{hnd}}, \text{pk}^{\text{hnd}}_{\text{tup}, \text{c}}, m_3^{\text{hnd}})$.
   33. **if** $x_1 = \text{payment} \land y_1 = \text{invoice} \land b_2 = b_3 = \text{true} \land \forall j = 2, \ldots, 5: x_j = y_j$ **then**
      34. **Output** (dispute, transfer, true, $x_2, x_3, x_4, x_5$) at $\text{PS}_\text{out}_{\text{tup}}$.
   35. **else**
      36. **Output** (dispute, transfer, false, $x_2, x_3, x_4, x_5$) at $\text{PS}_\text{out}_{\text{tup}}$.
   37. **end if**
38. **end if**