RZ 3565 (# 99575) 12/01/04
Computer Science 26 pages

Research Report

A Cryptographically Sound Dolev–Yao-Style Security Proof of an Electronic Payment System

Michael Backes and Markus Duermuth

IBM Research GmbH
Zurich Research Laboratory
8803 Rüschlikon
Switzerland
{mbc,due}@zurich.ibm.com

A Cryptographically Sound Dolev-Yao Style Security Proof of an Electronic Payment System

Michael Backes, Markus Duermuth
IBM Zurich Research Laboratory, Switzerland
{mbc,due}@zurich.ibm.com

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 [52, 51]. 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., [31, 29, 14]. The security of a cryptographic protocol is proved by reduction, 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.[1] They are almost always based on the so-called Dolev-Yao model [27], 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., [46, 42, 37, 17, 50, 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 [36, 16, 15, 38, 43, 10, 11]. 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 [13, 12] 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 [32], and efficient signature systems that are secure in this sense exist under reasonable assumptions [32, 26, 28].

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 [8]. 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 [21, 22, 25, 23, 24, 49]. Based on these works, a substantial body of commercial attempts at electronic payment systems emerged. The *i*KP family [13, 12] 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 [8] was restricted to passive adversaries and symmetric encryption [3, 2, 39]. Concurrently with [8], an extension to asymmetric encryption—but still under passive attacks only—was presented in [34]. The underlying Master's thesis [33] 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 [20]. The recent work of [45] gives a slightly more efficient implementation of asymmetric encryption than [8] (no additional tagging and randomization) at the cost of a much less general library and a weaker security notion. The outlook in [45] would essentially yield

[1] Efforts exist to formulate syntactic calculi for dealing with probabilism and polynomial-time considerations, in particular [47, 48, 35]. However, this approach cannot yet handle protocols with any degree of automation.

[8] again. Based on the ideal Dolev-Yao-style library, the well-known Needham-Schroeder-Lowe and Otway-Rees protocols have been analyzed in a cryptographically sound way [7, 5]. 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.

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 [53] and extended to asynchronous systems in [54, 19]. It extends the security notions of multiparty (one-step) function evaluation [55, 29, 30, 44, 9, 18] and the observational equivalence of [40]. 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., no party can cheat by tampering with these inputs. To simplify notation we let signatures include the signed message. We further assume that every participating party w initially holds a secret signature key sks_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 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, $\mathsf{receive}$, and allow from the client, the merchant, and the acquirer, respectively, and the outputs paid, $\mathsf{received}$, and $\mathsf{transfer}$ 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 $(\mathsf{receive}, d, p, u)$, which indicates agreement to receive the money p in exchange for the good d from u. The merchant computes a signature $sig_v := \mathsf{sign}_{sks_v}(\mathsf{invoice}, d, p, u, v)$ and sends $(\mathsf{invoice}, sig_v)$ to client u.

Upon receiving a message $(\mathsf{invoice}, sig_v)$, the client u tests if 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 (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 $sig_u := \mathsf{sign}_{sks_u}(\mathsf{payment}, d, p, u, v)$ and sends $(\mathsf{payment}, sig_u)$ to v.

Upon receiving a message $(\mathsf{payment}, sig_u)$, the merchant v tests if 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 sig_u for later use in disputes and sends $(\mathsf{auth\_request}, sig_u, sig_v)$ to the acquirer ac.

Upon receiving $(\mathsf{auth\_request}, sig_u, sig_v)$, the acquirer 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 ac has not yet received a local command $(\mathsf{allow}, d, p, u, v)$ indicating consent to the payment, he

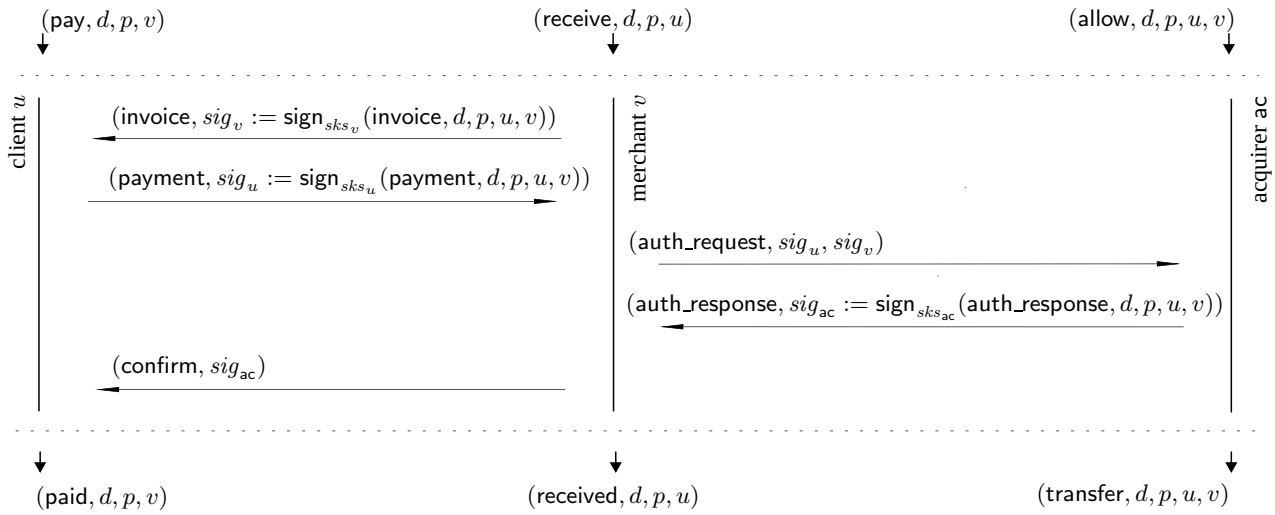

Figure 1: Optimistic Part of the Protocol

stores the received data until this local command occurs. If it has already occurred or when it occurs, ac computes $sig_{\mathsf{ac}} := \mathsf{sign}_{sks_{\mathsf{ac}}}(\mathsf{auth\_response}, d, p, u, v)$, sends $(\mathsf{auth\_response}, sig_{\mathsf{ac}})$ to v, and outputs $(\mathsf{transfer}, d, p, u, v)$ locally.

Upon receiving $(\mathsf{auth\_response}, sig_{\mathsf{ac}})$, the merchant v checks the validity of the signature with respect to 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 $(\mathsf{confirm}, sig_{\mathsf{ac}})$ to client u and outputs $(\mathsf{received}, d, p, u)$ locally.

Upon receiving $(\mathsf{confirm}, sig_{\mathsf{ac}})$, the client checks the validity of the signature with respect to ac's public key and if u has earlier sent a payment with matching data. He then outputs (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 ac) can start a dispute by inputting a local command $(\mathsf{dispute}, d, p, v)$ (or $(\mathsf{dispute}, d, p, u, v)$ if $w = \mathsf{ac}$). 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 $sig^* := \mathsf{sign}_{sks_w}(\mathsf{dispute}, sig_x, sig_{x'})$ where $\{x, x'\} = \{u, v, \mathsf{ac}\} \setminus \{w\}$ and sends sig^* to the trusted third party. Upon receiving a message 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 $(\mathsf{dispute}, \mathsf{true}, d, p, u, v)$, and $(\mathsf{dispute}, \mathsf{false}, d, p, u, v)$ otherwise.

3 The Payment Protocol Using the Dolev-Yao-style Cryptographic Library

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 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 [8], 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.

3.1 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 $\mathcal{M} := \{1, \ldots, n\}$ of users that is partitioned into a set $\mathcal{M}^{\mathsf{client}}$ of *clients*, a set $\mathcal{M}^{\mathsf{merchant}}$ of *merchants*, an *acquirer* ac, and a *trusted third party* ttp. By $\mathsf{M}_u^{\mathsf{PS}}$ we denote the payment protocol machine for a user u. Let Σ denote a finite alphabet and let Σ^* denote the set of strings over it.

3.1.1 Clients

Let $u \in \mathcal{M}^{\mathsf{client}}$ denote a client. The main data structure of $\mathsf{M}_u^{\mathsf{PS}}$ is a database D_u^{PS} 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 $pred$ involving attributes, $D[pred]$ means the subset of entries whose attributes fulfill $pred$. If $D[pred]$ contains only one element, we use the same notation for this element. Adding an entry x to D is abbreviated $D :\Leftarrow x$. Further, we write the list operation as $l := (x_1, \ldots, x_j)$, and the arguments are unambiguously retrievable as $l[i]$, with $l[i] = \downarrow$ if $i > j$. In our case, each entry x in D_u^{PS} can have the arguments

$$(ind, desc, price, merch, sig_{\mathsf{m}}, sig_{\mathsf{ac}}, status).$$

where the arguments have the following types and meaning:

- $x.ind \in \mathcal{INDS}$, called index, consecutively numbers all entries in D_u^{PS}. The set $\mathcal{INDS}$ 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 $D_u^{\mathsf{PS}}[i]$ for the selection $D_u^{\mathsf{PS}}[ind = i]$. We further use the convention that look-ups in D_u^{PS} always return the element with the smallest index whose attributes fulfill the queried predicate.
- $x.desc \in \Sigma^*$ is the description of the good to be purchased.
- $x.price \in \mathbb{N}$ denotes the price of the good.
- $x.merch \in \mathcal{M}^{\mathsf{merchant}}$ is the identifier of the merchant that should receive the payment.
- $x.sig_{\mathsf{m}}, x.sig_{\mathsf{ac}} \in \mathcal{HNDS}$ denote handles to the merchant's and the acquirer's signature, respectively. They will be stores during the execution of the protocol and read only for disputes. The set $\mathcal{HNDS}$ is yet another set isomorphic to $\mathbb{N}$. We always use a superscript "hnd" for handles.
- $x.status \in \{\mathsf{invoice}, \mathsf{pay}, \mathsf{processed}\}$ denotes the status of the transaction. Here $\mathsf{invoice}$ means that the client has received the invoice of the merchant, pay that the client gave consent to the payment, and $\mathsf{processed}$ that both events happened and that the payment has hence been performed.

Initially, D_u^{PS} is empty. $\mathsf{M}_u^{\mathsf{PS}}$ furthermore a variable $cur\_ind_u \in \mathcal{INDS}$ initialized with 0 counting the size of D_u^{PS}, and used as index for new entries in D_u^{PS}.

[2] For some frameworks there are compilers to generate these detailed protocol descriptions, e.g., [41]. This should be possible for this framework in a similar way.

Algorithm 1 Client: Evaluation of Users Inputs for Payment Consent in $\mathsf{M}_u^{\mathsf{PS}}$

Input: (pay, d, p, v) at $\mathsf{PS\_in}_u?$ with $d \in \Sigma^*$, $p \in \mathbb{N}$, and $v \in \mathcal{M}^{\mathsf{merchant}}$,

1: $i := D_u^{\mathsf{PS}}[desc = d \wedge price = p \wedge merch = v].ind.$
2: **if** $i \neq \downarrow \ \wedge D_u^{\mathsf{PS}}[i].status = \mathsf{invoice}$ **then**
3: $\quad D_u^{\mathsf{PS}}[i].status := \mathsf{processed}.$
4: $\quad \mathsf{payment}^{\mathsf{hnd}} \leftarrow \mathsf{store}(\mathsf{payment}).$
5: $\quad d^{\mathsf{hnd}} \leftarrow \mathsf{store}(d).$
6: $\quad p^{\mathsf{hnd}} \leftarrow \mathsf{store}(p).$
7: $\quad u^{\mathsf{hnd}} \leftarrow \mathsf{store}(u).$
8: $\quad v^{\mathsf{hnd}} \leftarrow \mathsf{store}(v).$
9: $\quad l^{\mathsf{hnd}} \leftarrow \mathsf{list}(\mathsf{payment}^{\mathsf{hnd}}, d^{\mathsf{hnd}}, p^{\mathsf{hnd}}, u^{\mathsf{hnd}}, v^{\mathsf{hnd}}).$
10: $\quad s^{\mathsf{hnd}} \leftarrow \mathsf{sign}(sks_u^{\mathsf{hnd}}, l^{\mathsf{hnd}}).$
11: $\quad m^{\mathsf{hnd}} \leftarrow \mathsf{list}(\mathsf{payment}^{\mathsf{hnd}}, s^{\mathsf{hnd}}).$
12: $\quad \mathsf{send\_i}(v, m^{\mathsf{hnd}}).$
13: **else if** $i = \downarrow$ **then**
14: $\quad D_u^{\mathsf{PS}} :\Leftarrow (cur\_ind_u\mathsf{++}, d, p, v, \downarrow, \downarrow, \mathsf{pay}).$
15: **end if**

The first type of input that $\mathsf{M}_u^{\mathsf{PS}}$ can receive is a message (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 $\mathsf{PS\_in}_u?$. 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. $\mathsf{M}_u^{\mathsf{PS}}$ first checks if a corresponding invoice with the same parameters has already been received before. In this case, the machine $\mathsf{M}_u^{\mathsf{PS}}$ 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_i means that $\mathsf{M}_u^{\mathsf{PS}}$ attempts to send the resulting term to v over an insecure channel. If no prior invoice message with suitable parameters occurred, $\mathsf{M}_u^{\mathsf{PS}}$ only creates a new database entry that will be processed when the invoice message is received. The superscript $^{\mathsf{hnd}}$ 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 [8] 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 $\mathsf{M}_v^{\mathsf{PS}}$ 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 $\mathsf{M}_u^{\mathsf{PS}}$ 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 $\mathsf{out}_u?$ and is of the form $(v, u, \mathsf{i}, l^{\mathsf{hnd}})$ where v is the supposed sender, i denotes that the channel is insecure, and l^{hnd} is a handle to a list. The port $\mathsf{out}_u?$ 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, $\mathsf{M}_u^{\mathsf{PS}}$ first determines if the message corresponds to an invoice message or a confirmation

Algorithm 2 Client: Evaluation of Inputs from the Cryptographic Library in $\mathsf{M}_u^{\mathsf{PS}}$

Input: $(v, u, \mathsf{i}, l^{\mathsf{hnd}})$ at out_u? with $v \in \mathcal{M}^{\mathsf{merchant}}$.

1: $l_j^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(l^{\mathsf{hnd}}, j)$ for $j = 1, 2$.
2: $l_1 \leftarrow \mathsf{retrieve}(l_1^{\mathsf{hnd}})$.
3: **if** $l_1 = \mathsf{invoice}$ **then**
4: $\quad m_2^{\mathsf{hnd}} \leftarrow \mathsf{msg\_of\_sig}(l_2^{\mathsf{hnd}})$.
5: $\quad b \leftarrow \mathsf{verify}(l_2^{\mathsf{hnd}}, pks_{u,v}^{\mathsf{hnd}}, m_2^{\mathsf{hnd}})$
6: $\quad x_j^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(m_2^{\mathsf{hnd}}, j)$ for $j = 1, \ldots, 5$.
7: $\quad x_j \leftarrow \mathsf{retrieve}(x_j^{\mathsf{hnd}})$ for $j = 1, \ldots, 5$.
8: $\quad i := D_u^{\mathsf{PS}}[desc = x_2 \wedge price = x_3 \wedge merch = x_5].ind$.
9: $\quad$ **if** $x_1 = \mathsf{invoice} \wedge x_4 = u \wedge x_5 = v \wedge b = \mathsf{true} \wedge D_u^{\mathsf{PS}}[i].status = \mathsf{pay}$ **then**
10: $\quad\quad D_u^{\mathsf{PS}}[i].sig_{\mathsf{m}} := l_2^{\mathsf{hnd}}$.
11: $\quad\quad D_u^{\mathsf{PS}}[i].status := \mathsf{processed}$.
12: $\quad\quad \mathsf{payment}^{\mathsf{hnd}} \leftarrow \mathsf{store}(\mathsf{payment})$.
13: $\quad\quad m_1^{\mathsf{hnd}} \leftarrow \mathsf{list}(\mathsf{payment}^{\mathsf{hnd}}, x_2^{\mathsf{hnd}}, x_3^{\mathsf{hnd}}, x_4^{\mathsf{hnd}}, x_5^{\mathsf{hnd}})$.
14: $\quad\quad s_1^{\mathsf{hnd}} \leftarrow \mathsf{sign}(sks_u^{\mathsf{hnd}}, m_1^{\mathsf{hnd}})$.
15: $\quad\quad m^{\mathsf{hnd}} \leftarrow \mathsf{list}(\mathsf{payment}^{\mathsf{hnd}}, s_1^{\mathsf{hnd}})$.
16: $\quad\quad \mathsf{send\_i}(v, m^{\mathsf{hnd}})$.
17: $\quad$ **else if** $x_1 = \mathsf{invoice} \wedge x_4 = u \wedge x_5 = v \wedge b = \mathsf{true} \wedge i = \downarrow$ **then**
18: $\quad\quad D_u^{\mathsf{PS}} :\Leftarrow (cur\_ind_u\mathsf{++}, x_2, x_3, x_5, l_2^{\mathsf{hnd}}, \downarrow, \mathsf{invoice})$.
19: $\quad$ **end if**
20: **else if** $l_1 = \mathsf{confirm}$ **then**
21: $\quad m_2^{\mathsf{hnd}} \leftarrow \mathsf{msg\_of\_sig}(l_2^{\mathsf{hnd}})$.
22: $\quad b \leftarrow \mathsf{verify}(l_2^{\mathsf{hnd}}, pks_{u,\mathsf{ac}}^{\mathsf{hnd}}, m_2^{\mathsf{hnd}})$
23: $\quad x_j^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(m_2^{\mathsf{hnd}}, j)$ for $j = 1, \ldots, 5$.
24: $\quad x_j \leftarrow \mathsf{retrieve}(x_j^{\mathsf{hnd}})$ for $j = 1, \ldots, 5$.
25: $\quad i := D_u^{\mathsf{PS}}[desc = x_2 \wedge price = x_3 \wedge merch = x_5 \wedge sig_{\mathsf{ac}} = \downarrow \wedge status = \mathsf{processed}].ind$.
26: $\quad$ **if** $x_1 = \mathsf{auth\_response} \wedge x_4 = u \wedge x_5 = v \wedge b = \mathsf{true} \wedge i \neq \downarrow$ **then**
27: $\quad\quad D_u^{\mathsf{PS}}[i].sig_{\mathsf{ac}} := l_2^{\mathsf{hnd}}$.
28: $\quad\quad$ Output $(\mathsf{paid}, x_2, x_3, x_5)$ at $\mathsf{PS\_out}_u$!.
29: $\quad$ **end if**
30: **end if**

Algorithm 3 Client: Evaluation of User Inputs for Disputes in $\mathsf{M}_u^{\mathsf{PS}}$

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

1: **if** $i := D_u^{\mathsf{PS}}[desc = d \wedge price = p \wedge merch = v \wedge sig_{\mathsf{m}} \neq \downarrow \wedge sig_{\mathsf{ac}} \neq \downarrow].ind \neq \downarrow$ **then**
2: $\quad \mathsf{dispute}^{\mathsf{hnd}} \leftarrow \mathsf{store}(\mathsf{dispute})$.
3: $\quad l^{\mathsf{hnd}} \leftarrow \mathsf{list}(\mathsf{dispute}^{\mathsf{hnd}}, D_u^{\mathsf{PS}}[i].sig_{\mathsf{m}}, D_u^{\mathsf{PS}}[i].sig_{\mathsf{ac}})$.
4: $\quad s^{\mathsf{hnd}} \leftarrow \mathsf{sign}(sks_u^{\mathsf{hnd}}, l^{\mathsf{hnd}})$.
5: $\quad m^{\mathsf{hnd}} \leftarrow \mathsf{list}(s^{\mathsf{hnd}})$.
6: $\quad \mathsf{send\_i}(\mathsf{ttp}, m^{\mathsf{hnd}})$.
7: **end if**

message, i.e., if the message could correspond to the first or the fifth message in the protocol description. In the first case, $\mathsf{M}_u^{\mathsf{PS}}$ 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, $\mathsf{M}_u^{\mathsf{PS}}$ 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, $\mathsf{M}_u^{\mathsf{PS}}$ only creates a new database entry that contains the payment information of the invoice message. The evaluation of a confirmation message works similarly: $\mathsf{M}_u^{\mathsf{PS}}$ 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 $\mathsf{PS\_out}_u!$ the successful completion of the payment.

Finally, $\mathsf{M}_u^{\mathsf{PS}}$ can receive a dispute message $(\mathsf{dispute}, d, p, v)$ from its user at $\mathsf{PS\_in}_u?$. The behaviour of $\mathsf{M}_u^{\mathsf{PS}}$ 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, $\mathsf{M}_u^{\mathsf{PS}}$ builds up a term according to the protocol description and sends it to the trusted third party.

Every algorithms 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 $\mathsf{M}_u^{\mathsf{PS}}$ for $u \in \mathcal{M}$ receives $\downarrow$ as the answer of the cryptographic library to a command, then $\mathsf{M}_u^{\mathsf{PS}}$ aborts the execution of the current algorithm, except for the command types* list_proj *or* send_i.

3.1.2 Merchants

Let $u \in \mathcal{M}^{\mathsf{merchant}}$ denote a merchant. Similar to the protocol machines of the clients, the machine $\mathsf{M}_u^{\mathsf{PS}}$ maintains an initially empty database D_u^{PS} as its main data structure, together with a variable $cur\_ind_u \in \mathcal{INDS}$ for counting the size of the database and creating new indices. The entries of D_u^{PS} have the form

$$(ind, desc, price, client, sig_{\mathsf{c}}, sig_{\mathsf{ac}}).$$

For $x \in D_u^{\mathsf{PS}}$:

- $x.ind, x.desc, x.price$, and $x.sig_{\mathsf{ac}}$ are defined as in the database of the client machines.
- $x.client \in \mathcal{M}^{\mathsf{client}}$ denotes the client of this transaction.
- $x.sig_{\mathsf{c}} \in \mathcal{HNDS}$ denotes the handle of the client's signature, which will be collected during the protocol execution.

The machine $\mathsf{M}_u^{\mathsf{PS}}$ accepts inputs for sending an invoice to a client and for initiating a dispute from its user at port $\mathsf{PS\_in}_u?$, and inputs from the cryptographic library at port $\mathsf{in}_u?$. After explaining the behavior of the protocol machines of the clients, the behavior of $\mathsf{M}_u^{\mathsf{PS}}$ 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 $\mathsf{M}_{\mathsf{ac}}^{\mathsf{PS}}$ of the acquirer maintains a variable $cur\_ind_{\mathsf{ac}} \in \mathcal{INDS}$ initialized with 0 and an initially empty database $D_{\mathsf{ac}}^{\mathsf{PS}}$, where each entry $x \in D_{\mathsf{ac}}^{\mathsf{PS}}$ can have arguments

$$(ind, desc, price, client, merch, sig_{\mathsf{c}}, sig_{\mathsf{m}}, status)$$

with the following types and meanings:

- $x.ind, x.desc, x.price, x.client, x.merch, x.sig_{\mathsf{c}}$, and $x.sig_{\mathsf{m}}$ are defined as in the databases of the client and merchant machines.
- $x.status \in \{\mathsf{allow}, \mathsf{auth\_request}, \mathsf{processed}\}$ 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 $\mathsf{M}_{\mathsf{ac}}^{\mathsf{PS}}$ accepts inputs for processing a payment from user ac at port $\mathsf{PS\_in}_{\mathsf{ac}}?$, and inputs from the cryptographic library at port $\mathsf{in}_{\mathsf{ac}}?$. The algorithmic description of $\mathsf{M}_{\mathsf{ac}}^{\mathsf{PS}}$ is postponed to the Appendix.

3.1.4 Trusted Third Party

The machine $\mathsf{M}_{\mathsf{ttp}}^{\mathsf{PS}}$ of the trusted third party only accepts input from the cryptographic library at port $\mathsf{in}_{\mathsf{ttp}}?$. Upon receiving such an input, it first checks the validity of the signature and if the message in indeed a well-formed dispute message. It then identifies the parties that are involved in the payment and determine 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 $\mathsf{M}_u^{\mathsf{PS}}$ already has a handle sks_u^{hnd} to its own secret signature key and handles $pks_{u,v}^{\mathsf{hnd}}$ 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 $\mathsf{M}_u^{\mathsf{PS}}$ 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 [54]. 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 $\mathsf{M}_u^{\mathsf{PS}}$ 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 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 $\mathcal{H}$ of honest participants, the user interface of the ideal and real cryptographic library is the port set $S_{\mathcal{H}}^{\mathsf{cry}} := \{\mathsf{in}_u?, \mathsf{out}_u! \mid u \in \mathcal{H}\}$. 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 $\mathsf{TH}_{\mathcal{H}}^{\mathsf{cry}}$ 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 $\hat{M}_{\mathcal{H}}^{\mathsf{cry}} := \{\mathsf{M}_{u,\mathcal{H}}^{\mathsf{cry}} \mid u \in \mathcal{H}\}$ of real cryptographic machines. The corresponding systems are called $Sys^{\mathsf{cry,id}} := \{(\{\mathsf{TH}_{\mathcal{H}}^{\mathsf{cry}}\}, S_{\mathcal{H}}^{\mathsf{cry}}) \mid \mathcal{H} \subseteq \{1, \ldots, n\}\}$ and $Sys^{\mathsf{cry,real}} := \{(\hat{M}_{\mathcal{H}}^{\mathsf{cry}}, S_{\mathcal{H}}^{\mathsf{cry}}) \mid \mathcal{H} \subseteq \{1, \ldots, n\}\}$.

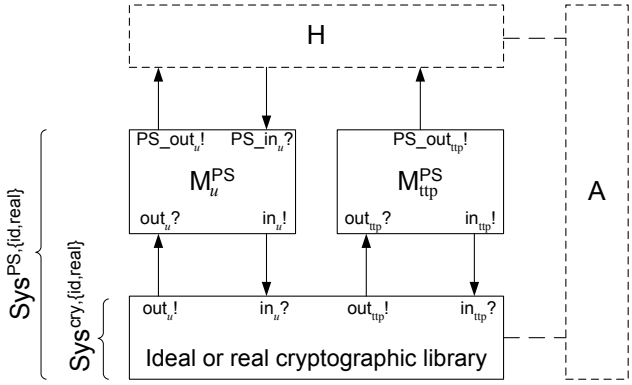

Figure 2: Overview of the Payment System (here for the case $\mathcal{H} = \{u, \mathsf{ttp}\}$).

The user interface of the payment protocol machines is $S_{\mathcal{H}}^{\mathsf{PS}} := \{\mathsf{PS\_in}_u?, \mathsf{PS\_out}_u! \mid u \in \mathcal{H} \setminus \{\mathsf{ttp}\}\} \cup \{\mathsf{PS\_out}_{\mathsf{ttp}}!\}$, cf. Figure 2. The ideal and real payment systems serving this interface differ only in the cryptographic library. With $\hat{M}_{\mathcal{H}}^{\mathsf{PS}} := \{\mathsf{M}_u^{\mathsf{PS}} \mid u \in \mathcal{H}\}$, they are $Sys^{\mathsf{PS,id}} := \{(\hat{M}_{\mathcal{H}}^{\mathsf{PS}} \cup \{\mathsf{TH}_{\mathcal{H}}^{\mathsf{cry}}\}, S_{\mathcal{H}}^{\mathsf{PS}}) \mid \{\mathsf{ttp}\} \subseteq \mathcal{H} \subseteq \{1, \ldots, n\}\}$ and $Sys^{\mathsf{PS,real}} := \{(\hat{M}_{\mathcal{H}}^{\mathsf{PS}} \cup \hat{M}_{\mathcal{H}}^{\mathsf{cry}}, S_{\mathcal{H}}^{\mathsf{PS}}) \mid \{\mathsf{ttp}\} \subseteq \mathcal{H} \subseteq \{1, \ldots, n\}\}$.

3.4 On Polynomial Runtime

In order to be valid users of the real cryptographic library, the machines $\mathsf{M}_u^{\mathsf{PS}}$ have to be polynomial-time. We therefore define that each machine $\mathsf{M}_u^{\mathsf{PS}}$ 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 systems is that no money can be transfered without the client's consent. This can be captured as an *integrity property* in the underlying model which are formally sets of traces at the user interfaces of a system, i.e., here at the port sets $S_{\mathcal{H}}^{\mathsf{PS}}$. Intuitively, an integrity property 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 $(\mathsf{transfer}, d, p, u, v)$ at a port $\mathsf{PS\_out}_{\mathsf{ac}}?$ for an honest client u, an honest acquirer ac, and an arbitrary (potentially malicious) merchant v is preceded by an input (pay, d, p, v) at $\mathsf{PS\_in}_u?$. 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 $\mathsf{SuccessHonestTerm}(d, p, u, v, \mathsf{ac}, r, t)$ 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 $(u \in \mathcal{H} \wedge \mathsf{PS\_out}_u!(\mathsf{paid}, d, p, v) \in r_t)$, $(v \in \mathcal{H} \wedge \mathsf{PS\_out}_v!(\mathsf{received}, d, p, u) \in r_t)$, and $(\mathsf{ac} \in \mathcal{H} \wedge \mathsf{PS\_out}_{\mathsf{ac}}!(\mathsf{transfer}, d, p, u, v) \in r_t)$.

Definition 4.1 (Weak Atomicity) *A trace r is contained in $Req^{\mathsf{weak\_atom}}$ if and only if for all $d \in \Sigma^*$, $p \in \mathbb{N}$,*

$u \in \mathcal{M}^{\mathsf{client}}$, $v \in \mathcal{M}^{\mathsf{merchant}}$, *and* $t_2 \in \mathbb{N}$*:*

$$\begin{aligned}&\mathsf{SuccessHonestTerm}(d,p,u,v,\mathsf{ac},r,t_2) \Rightarrow \exists t_1, t_1', t_1'' < t_2 \colon \\ &\Big((u \in \mathcal{H} \Rightarrow \mathsf{PS\_in}_u?(\mathsf{pay},d,p,v) \in r_{t_1}) \wedge (v \in \mathcal{H} \Rightarrow \mathsf{PS\_in}_v?(\mathsf{receive},d,p,u) \in r_{t_1'}) \\ &\quad \wedge (\mathsf{ac} \in \mathcal{H} \Rightarrow \mathsf{PS\_in}_{\mathsf{ac}}?(\mathsf{allow},d,p,u,v) \in r_{t_1''})\Big).\end{aligned}$$

$\diamond$

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 $Req^{\mathsf{corr\_disp\_client}}$ if and only if for all $d \in \Sigma^*$, $p \in \mathbb{N}$, $u \in \mathcal{H}^{\mathsf{client}}$, $v \in \mathcal{M}^{\mathsf{merchant}}$, and $t_3 \in \mathbb{N}$:*

$$\begin{aligned}&\mathsf{PS\_out}_{\mathsf{ttp}}!(\mathsf{dispute},\mathsf{paid},\mathsf{false},d,p,u,v) \in r_{t_3} \\ \Rightarrow\ &\exists t_2 < t_3 \colon (\mathsf{PS\_in}_u?(\mathsf{dispute},d,p,v) \in r_{t_2} \wedge \forall t_1 < t_2 \colon \mathsf{PS\_out}_u!(\mathsf{paid},d,p,v) \notin r_{t_1}).\end{aligned}$$

$\diamond$

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 $Req^{\mathsf{no\_frame\_client}}$ if and only if for all $d \in \Sigma^*$, $p \in \mathbb{N}$, $u \in \mathcal{H}^{\mathsf{client}}$, $v \in \mathcal{M}^{\mathsf{merchant}}$, $t_2 \in \mathbb{N}$, and $x \in \{\mathsf{received},\mathsf{transfer}\}$:*

$$\mathsf{PS\_out}_{\mathsf{ttp}}?(\mathsf{dispute},x,\mathsf{true},d,p,u,v) \in r_{t_2} \Rightarrow \exists t_1 < t_2 \colon \mathsf{PS\_in}_u?(\mathsf{pay},d,p,v) \in r_{t_1}.$$

$\diamond$

Let $Req^{\mathsf{corr\_disp}}$ denote the conjunction of $Req^{\mathsf{corr\_disp\_client}}$ and the corresponding properties for merchants and the acquirer; similarly, let $Req^{\mathsf{no\_frame}}$ denote the conjunction of $Req^{\mathsf{no\_frame\_client}}$ and its counterparts for merchants and the acquirer.

The notion of a system Sys fulfilling an integrity property Req essentially comes in two flavors [6]. *Perfect fulfillment,* $Sys \models^{\mathsf{perf}} Req$, means that the integrity property holds for all traces arising in runs of Sys (a well-defined notion from the underlying model [54]). *Computational fulfillment,* $Sys \models^{\mathsf{poly}} Req$, 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 $Req^{\mathsf{PS}} := Req^{\mathsf{weak\_atom}} \cap Req^{\mathsf{corr\_disp}} \cap Req^{\mathsf{no\_frame}}$. For the payment system from Section 3.3 we have $Sys^{\mathsf{PS,id}} \models^{\mathsf{perf}} Req^{\mathsf{PS}}$ and $Sys^{\mathsf{PS,real}} \models^{\mathsf{poly}} Req^{\mathsf{PS}}$.* $\square$

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. The long version of this paper contains this modeling and the corresponding proofs in detail.

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 Sys_1 securely implements another system Sys_2 in called reactive simulatability (recall the introduction), and is written $Sys_1 \geq_{\mathsf{sec}}^{\mathsf{poly}} Sys_2$ (in the computational case). The main result of [8] is therefore

$$Sys^{\mathsf{cry,real}} \geq_{\mathsf{sec}}^{\mathsf{poly}} Sys^{\mathsf{cry,id}}. \tag{1}$$

Since $Sys^{\mathsf{PS,real}}$ and $Sys^{\mathsf{PS,id}}$ are compositions of the same protocol with $Sys^{\mathsf{cry,real}}$ and $Sys^{\mathsf{cry,id}}$, respectively, the composition theorem of [54] and (1) imply

$$Sys^{\mathsf{PS,real}} \geq_{\mathsf{sec}}^{\mathsf{poly}} Sys^{\mathsf{PS,id}}. \tag{2}$$

Showing the theorem's preconditions is easy since the machines $\mathsf{M}_u^{\mathsf{PS}}$ are polynomial-time (see Section 3.4). Finally, the integrity preservation theorem from [6] and (2) imply for every integrity property Req that

$$(Sys^{\mathsf{PS,id}} \models^{\mathsf{poly}} Req) \;\Rightarrow\; (Sys^{\mathsf{PS,real}} \models^{\mathsf{poly}} Req). \tag{3}$$

Hence if we prove $Sys^{\mathsf{PS,id}} \models^{\mathsf{perf}} Req^{\mathsf{PS}}$, we immediately obtain $Sys^{\mathsf{PS,real}} \models^{\mathsf{poly}} Req^{\mathsf{PS}}$ 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 Req^{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 briefly explain how to exploit them to prove the overall integrity property of the payment system. Proving the integrity property in fact turns out to be easy once the invariants have been established; we postpone this proof and the proof of the invariants to Appendix B, preceded by a detailed description of the state transitions of the ideal cryptographic library as needed in these proofs in Appendix A.

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 $u = \mathsf{a}$ 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 sks. There are multiple ways to deal with this in prior Dolev-Yao models, e.g., counting (for nonces) and multisets. The version in [8] 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 $Sys^{\mathsf{cry,id}}$, but we omitted them for readability.

Similarly, n and a tuple L' should be parameters of our ideal payment system $Sys^{\mathsf{PS,id}}$, see Section 3.4. As the machines $\mathsf{M}_u^{\mathsf{PS}}$ 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 $\mathsf{TH}_{\mathcal{H}}^{\mathsf{cry}}$ for every set $\mathcal{H}$ 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 [8] is a database D in which each entry x in D can have the arguments

$$(ind, type, arg, hnd_{u_1}, \ldots, hnd_{u_m}, hnd_{\mathsf{a}}, len),$$

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

- $x.ind$ is the global index of an entry.
- $x.type \in typeset$ 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.
- $x.arg = (a_1, a_2, \ldots, a_j)$ is a possibly empty list of arguments. Arguments of type $\mathcal{IND}\mathcal{S}$ are indices of other entries (subterms); we sometimes distinguish them by a superscript "ind".
- $x.hnd_u \in \mathcal{HND}\mathcal{S} \cup \{\downarrow\}$ for $u \in \mathcal{H} \cup \{\mathsf{a}\}$ are handles, where $x.hnd_u = \downarrow$ means that u does not know this entry.
- $x.len \in \mathbb{N}_0$ denotes the length of the entry.

The machine $\mathsf{TH}_{\mathcal{H}}^{\mathsf{cry}}$ has a counter $size \in \mathcal{IND}\mathcal{S}$ for the current size of D and counters $curhnd_u$ (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 $\{u_1, \ldots, u_m\} := \mathcal{H}$:

$$\begin{aligned}
&(sks_u, type := \mathsf{sks}, arg := (0), hnd_u := sks_u^{\mathsf{hnd}}, len := 0);\\
&(pks_u, type := \mathsf{pks}, arg := (), hnd_{u_1} := pks_{u_1,u}^{\mathsf{hnd}}, \ldots, hnd_{u_m} := pks_{u_m,u}^{\mathsf{hnd}},\\
&\qquad hnd_{\mathsf{a}} := pks_{\mathsf{a},u}^{\mathsf{hnd}}, len := \mathsf{pks_len}^*(k)).
\end{aligned}$$

Here sks_u and pks_u are consecutive natural numbers, $\mathsf{pks\_len}^*$ 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 [8] and will not matter in the sequel.

6.2 Invariants

This section contains invariants of the system $Sys^{\mathsf{PS,id}}$, 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 $\mathsf{processed}$. We let $r_t : D_u^{\mathsf{PS}}$ denote the contents of database D_u^{PS} at time t in trace r.

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

$$\begin{aligned} & r_{t_1} : D_u^{\mathsf{PS}}[i].x \neq \downarrow \wedge x \neq status \Rightarrow \forall t_2 > t_1 \colon (r_{t_2} : D_u^{\mathsf{PS}}[i].x = r_{t_1} : D_u^{\mathsf{PS}}[i].x) \\ \wedge \quad & s \in \{\mathsf{pay}, \mathsf{invoice}, \mathsf{allow}, \mathsf{auth\_request}\} \Rightarrow \forall t_2 > t_1 \colon (r_{t_2} : D_u^{\mathsf{PS}}[i].status \in \{s, \mathsf{processed}\}). \end{aligned}$$

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 $u \in \mathcal{H}$, $i_1, i_2 \in \mathcal{INDS}$, $t \in \mathbb{N}$, and all traces r arising in runs of $Sys^{\mathsf{PS,id}}$:*

$$\begin{aligned} & r_t : D_u^{\mathsf{PS}}[i_1].price = r_t : D_u^{\mathsf{PS}}[i_2].price \neq \downarrow \wedge r_t : D_u^{\mathsf{PS}}[i_1].desc = D_u^{\mathsf{PS}}[i_2].desc \neq \downarrow \wedge \\ & r_t : D_u^{\mathsf{PS}}[i_1].merch = D_u^{\mathsf{PS}}[i_2].merch \neq \downarrow \Rightarrow (i_1 = i_2). \end{aligned}$$

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 $\mathsf{payment}$ element are of the form $(\mathsf{payment}, d, p, u, v)$, and that a local command (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.

Invariant 3 *(Correct Signing (Client Part)) For all $u \in \mathcal{M}^{\mathsf{client}} \cap \mathcal{H}$, $i \in \mathcal{INDS}$, $t_2 \in \mathbb{N}$, and all traces r arising in runs of $Sys^{\mathsf{PS,id}}$:*

$$\begin{aligned} & r_{t_2} : D[i].type = \mathsf{sig} \wedge r_{t_2} : D[i].arg[1] = pks_u \wedge x_1 = \mathsf{payment} && \text{\# If a } \mathsf{payment} \text{ message is} \\ & \Rightarrow && \text{\# signed with } u\text{'s key, then} \\ & \Big(x_4 = u \wedge \forall j = 1, \ldots, 5 \colon (r_{t_2} : D[y_j].type = \mathsf{data}) \wedge && \text{\# the message is of the correct format} \\ & \exists t_1 < t_2 \colon (\mathsf{PS\_in}_u?(\mathsf{pay}, x_2, x_3, x_5) \in r_{t_1}) \Big), && \text{\# and a matching input has occurred.} \end{aligned}$$

where $l := r_{t_2} : D[i].arg[2]$ denotes the index of the signed list, $y_j := r_{t_2} : D[l].arg[j]$ the indices of the list elements, and $x_j := r_{t_2} : D[y_j].arg[1]$ the actual data for $j = 1, \ldots, 5$.

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., sig_{m} is a signature of a list $(\mathsf{invoice}, d, p, u, v)$ signed by v and sig_{ac} is a signature of a list $(\mathsf{auth\_response}, d, p, u, v)$ signed by ac.

Invariant 4 *(Correct Storing (Client Part)) For all $u \in \mathcal{M}^{\mathsf{client}} \cap \mathcal{H}$, $d \in \Sigma^*$, $p \in \mathbb{N}$, $v \in \mathcal{M}^{\mathsf{merchant}}$, $t \in \mathbb{N}$, and all traces r arising in runs of $Sys^{\mathsf{PS,id}}$:*

$$
\begin{aligned}
&\mathsf{PS\_out}_u!(\mathsf{paid}, d, p, v) \in r_t \Rightarrow && \text{\# If } u \text{ terminates the protocol, then}\\
&\Big(r_t : D[hnd_u = s^{\mathsf{m}}].type = \mathsf{sig} \wedge r_t : D[pk^{\mathsf{m}}].hnd_u = pks^{\mathsf{hnd}}_{u,v} \wedge && \text{\# the signatures of the merchant and}\\
&r_t : D[hnd_u = s^{\mathsf{ac}}].type = \mathsf{sig} \wedge r_t : D[pk^{\mathsf{ac}}].hnd_u = pks^{\mathsf{hnd}}_{u,\mathsf{ac}} \wedge && \text{\# of the acquirer have been stored,}\\
&x_1^{\mathsf{m}} = \mathsf{invoice} \wedge x_2^{\mathsf{m}} = d \wedge x_3^{\mathsf{m}} = p \wedge x_5^{\mathsf{m}} = v \wedge && \text{\# and they range}\\
&x_1^{\mathsf{ac}} = \mathsf{auth\_response} \wedge x_2^{\mathsf{ac}} = d \wedge x_3^{\mathsf{ac}} = p \wedge x_5^{\mathsf{ac}} = v\Big), && \text{\# over the correct data.}
\end{aligned}
$$

where for $w \in \{\mathsf{m}, \mathsf{ac}\}$, we let $s^w := r_t : D_u^{\mathsf{PS}}[desc = d \wedge price = p \wedge merch = v].sig_w$ denote the handle to w's signature, $pk^w := r_t : D[hnd_u = s^w].arg[1]$ the index of the public key used, $l^w := r_t : D[hnd_u = s^w].arg[2]$ the index of the signed list, $y_j^w := r_t : D[l^w].arg[j]$ the indices of the list elements, and $x_j^w := r_t : D[y_j^w].arg[1]$ the actual data for $j = 1, \ldots, 5$.

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. 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 a such a proof for protocols involving digital signatures. 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. We therefore hope that our hand-made proof helps to pave the way towards automated, cryptographically sound proofs of electronic payment systems and many other security protocols.

References

[1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[2] M. Abadi and J. Jürjens. Formal eavesdropping and its computational interpretation. In *Proc. 4th International Symposium on Theoretical Aspects of Computer Software (TACS)*, pages 82–94, 2001.

[3] M. Abadi and P. Rogaway. Reconciling two views of cryptography: The computational soundness of formal encryption. In *Proc. 1st IFIP International Conference on Theoretical Computer Science*, volume 1872 of *LNCS*, pages 3–22. Springer, 2000.

[4] N. Asokan, P. Janson, M. Steiner, and M. Waidner. State of the art in electronic payment systems. *Advances in Computers*, 43:425–449, 2000.

[5] M. Backes. A cryptographically sound Dolev-Yao style security proof of the Otway-Rees protocol. In *Proc. 9th European Symposium on Research in Computer Security (ESORICS)*, volume 3193 of *LNCS*, pages 89–108. Springer, 2004.

[6] M. Backes and C. Jacobi. Cryptographically sound and machine-assisted verification of security protocols. In *Proc. 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2607 of *LNCS*, pages 675–686. Springer, 2003.

[7] M. Backes and B. Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. In *Proc. 23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 1–12, 2003.

[8] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations (extended abstract). In *Proc. 10th ACM Conference on Computer and Communications Security*, pages 220–230, 2003. Full version in IACR Cryptology ePrint Archive 2003/015, Jan. 2003, `http://eprint.iacr.org/`.

[9] D. Beaver. Secure multiparty protocols and zero knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):75–122, 1991.

[10] G. Bella, F. Massacci, L. C. Paulson, and P. Tramontano. Formal verification of cardholder registration in SET. In *Proc. 6th European Symposium on Research in Computer Security (ESORICS)*, volume 1895 of *LNCS*, pages 159–174. Springer, 2000.

[11] G. Bella, L. C. Paulson, and F. Massacci. The verification of an industrial payment protocol: the SET purchase phase. In *Proc. 9th ACM Conference on Computer and Communications Security*, pages 12–20, 2002.

[12] M. Bellare, J. A. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, E. V. Herreweghen, and M. Waidner. Design, implementation and deployment of the iKP secure electronic payment system. *IEEE Journal on Selected Areas in Communications*, 18(4):611–627, 2000.

[13] M. Bellare, J. A. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, and M. Waidner. iKP - A family of secure electronic payment protocols. In *Proceedings of the First Usenix Workshop on Electronic Commerce*, 1995.

[14] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology: CRYPTO '93*, volume 773 of *LNCS*, pages 232–249. Springer, 1994.

[15] D. Bolignano. Towards the formal verification of electronic commerce protocols. In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, pages 133–146, 1997.

[16] S. Brackin. Automatic formal analyses of two large commercial protocols. In *Proc. DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[17] M. Burrows, M. Abadi, and R. Needham. A logic for authentication. Technical Report 39, SRC DIGITAL, 1990.

[18] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 3(1):143–202, 2000.

[19] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001. Extended version in Cryptology ePrint Archive, Report 2000/67, `http://eprint.iacr.org/`.

[20] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. In *Proc. 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 209–218, 1998.

[21] D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: CRYPTO'82*, pages 199–203, 1983.

[22] D. Chaum. Security without identification: Transaction systems to make Big Brother obsolete. *Communications of the ACM*, 28:1030–1044, 1985.

[23] D. Chaum. Online cash checks. In *Advances in Cryptology: EUROCRYPT'89*, volume 434 of *LNCS*, pages 288–293, 1989.

[24] D. Chaum, B. den Boer, E. van Heyst, S. F. Mjolsnes, and A. Steenbeek. Efficient offline electronic checks (extended abstract). In *Advances in Cryptology: EUROCRYPT'89*, volume 434 of *LNCS*, pages 294–301, 1989.

[25] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *Advances in Cryptology: CRYPTO'88*, volume 403 of *LNCS*, pages 319–327, 1988.

[26] R. Cramer and I. Damgård. New generation of secure and practical RSA-based signatures. In *Advances in Cryptology: CRYPTO '96*, volume 1109 of *LNCS*, pages 173–185. Springer, 1996.
[27] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
[28] C. Dwork and M. Naor. An efficient existentially unforgeable signature scheme and its applications. *Journal of Cryptology*, 11(3):187–208, 1998.
[29] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game – or – a completeness theorem for protocols with honest majority. In *Proc. 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.
[30] S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority. In *Advances in Cryptology: CRYPTO '90*, volume 537 of *LNCS*, pages 77–93. Springer, 1990.
[31] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.
[32] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.
[33] J. Herzog. Computational soundness of formal adversaries. Master Thesis, 2002.
[34] J. Herzog, M. Liskov, and S. Micali. Plaintext awareness via key registration. In *Advances in Cryptology: CRYPTO 2003*, volume 2729 of *LNCS*, pages 548–564. Springer, 2003.
[35] R. Impagliazzo and B. M. Kapron. Logics for reasoning about cryptographic constructions. In *Proc. 44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 372–381, 2003.
[36] R. Kailar. Reasoning about accountability in protocols for electronic commerce. In *Proc. 16th IEEE Symposium on Security & Privacy*, pages 236–250, 1995.
[37] R. Kemmerer. Analyzing encryption protocols using formal verification techniques. *IEEE Journal on Selected Areas in Communications*, 7(4):448–457, 1989.
[38] V. Kessler and H. Neumann. A sound logic for analysing electronic commerce protocols. In *Proc. 5th European Symposium on Research in Computer Security (ESORICS)*, volume 1485 of *LNCS*, pages 345–360. Springer, 1998.
[39] P. Laud. Semantics and program analysis of computationally secure information flow. In *Proc. 10th European Symposium on Programming (ESOP)*, pages 77–91, 2001.
[40] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proc. 5th ACM Conference on Computer and Communications Security*, pages 112–121, 1998.
[41] G. Lowe. Casper: A compiler for the analysis of security protocols. In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, pages 18–30, 1997.
[42] C. Meadows. Using narrowing in the analysis of key management protocols. In *Proc. 10th IEEE Symposium on Security & Privacy*, pages 138–147, 1989.
[43] C. Meadows and P. Syverson. A formal specification of requirements for payment transactions in the SET protocol. In *Proc. 2nd Financial Cryptography Conference (FC)*, volume 1465 of *LNCS*, pages 122–140. Springer, 1998.
[44] S. Micali and P. Rogaway. Secure computation. In *Advances in Cryptology: CRYPTO '91*, volume 576 of *LNCS*, pages 392–404. Springer, 1991.
[45] D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proc. 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *LNCS*, pages 133–151. Springer, 2004.
[46] J. K. Millen. The interrogator: A tool for cryptographic protocol security. In *Proc. 5th IEEE Symposium on Security & Privacy*, pages 134–141, 1984.
[47] J. Mitchell, M. Mitchell, and A. Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *Proc. 39th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 725–733, 1998.
[48] J. Mitchell, M. Mitchell, A. Scedrov, and V. Teague. A probabilistic polynominal-time process calculus for analysis of cryptographic protocols (preliminary report). *Electronic Notes in Theoretical Computer Science*, 47:1–31, 2001.
[49] T. Okamoto and K. Ohta. Universal electronic cash. In *Advances in Cryptology: CRYPTO '91*, volume 576 of *LNCS*, pages 324–337. Springer, 1992.
[50] L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Cryptology*, 6(1):85–128, 1998.

[51] B. Pfitzmann, M. Schunter, and M. Waidner. How to break another "provably secure" payment system. In *Advances in Cryptology: EUROCRYPT '95*, volume 921 of *LNCS*, pages 121–132. Springer, 1995.
[52] B. Pfitzmann and M. Waidner. How to break and repair a "provably secure" untraceable payment system. In *Advances in Cryptology: CRYPTO '91*, volume 576 of *LNCS*, pages 338–350. Springer, 1992.
[53] B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *Proc. 7th ACM Conference on Computer and Communications Security*, pages 245–254, 2000.
[54] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. 22nd IEEE Symposium on Security & Privacy*, pages 184–200, 2001. Extended version of the model (with Michael Backes) IACR Cryptology ePrint Archive 2004/082.
[55] A. C. Yao. Protocols for secure computations. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 160–164, 1982.

A Command Evaluation by the Ideal Cryptographic Library

This section contains the definition of the cryptographic commands used for modeling the investigated payment protocol, and the local adversary commands that model the extended capabilities of the adversary as far as needed to prove the invariants. Recall that we deal with top levels of Dolev-Yao-style terms, and that commands typically create a new term with its index, type, arguments, handles, and length functions, or parse an existing term. We present the full definitions of the commands, but the reader can ignore the length functions, which have names x_len. By $x := y\mathsf{++}$ for integer variables x, y we mean $y := y + 1; x := y$. The length of a message m is denoted as $\mathsf{len}(m)$.

Each input c at a port $\mathsf{in}_u?$ with $u \in \mathcal{H} \cup \{\mathsf{a}\}$ should be a list $(cmd, x_1, \ldots, x_j)$ with cmd from a fixed list of commands and certain parameter domains. We usually write it $y \leftarrow cmd(x_1, \ldots, x_j)$ with a variable y designating the result that $\mathsf{TH}^{\mathsf{cry}}_{\mathcal{H}}$ returns at $\mathsf{out}_u!$. The algorithm $i^{\mathsf{hnd}} := \mathsf{ind2hnd}_u(i)$ (with side effect) denotes that $\mathsf{TH}^{\mathsf{cry}}_{\mathcal{H}}$ determines a handle i^{hnd} for user u to an entry $D[i]$: If $i^{\mathsf{hnd}} := D[i].hnd_u \neq \downarrow$, it returns that, else it sets and returns $i^{\mathsf{hnd}} := D[i].hnd_u := curhnd_u\mathsf{++}$. On non-handles, it is the identity function. The function $\mathsf{ind2hnd}^*_u$ applies $\mathsf{ind2hnd}_u$ to each element of a list.

In the following definitions, we assume that a cryptographic command is input at port $\mathsf{in}_u?$ with $u \in \mathcal{H} \cup \{\mathsf{a}\}$. First, we describe the commands for storing and retrieving data via handles.

- *Storing:* $m^{\mathsf{hnd}} \leftarrow \mathsf{store}(m)$, for $m \in \{0,1\}^{\mathsf{max\_len}(k)}$.

  If $i := D[type = \mathsf{data} \wedge arg = (m)].ind \neq \downarrow$ then return $m^{\mathsf{hnd}} := \mathsf{ind2hnd}_u(i)$. Otherwise if $\mathsf{data\_len}^*(\mathsf{len}(m)) > \mathsf{max\_len}(k)$ return $\downarrow$. Else set $m^{\mathsf{hnd}} := curhnd_u\mathsf{++}$ and

  $$D \ :\Leftarrow \ (ind := size\mathsf{++}, type := \mathsf{data}, arg := (m), hnd_u := m^{\mathsf{hnd}}, len := \mathsf{data\_len}^*(\mathsf{len}(m))).$$

- *Retrieval:* $m \leftarrow \mathsf{retrieve}(m^{\mathsf{hnd}})$.

  $m := D[hnd_u = m^{\mathsf{hnd}} \wedge type = \mathsf{data}].arg[1]$.

Next we describe list creation and projection. Lists cannot include secret keys of the public-key systems (entries of type ske, sks) because no information about those must be given away.

- *Generate a list:* $l^{\mathsf{hnd}} \leftarrow \mathsf{list}(x_1^{\mathsf{hnd}}, \ldots, x_j^{\mathsf{hnd}})$, for $0 \leq j \leq \mathsf{max\_len}(k)$.

  Let $x_i := D[hnd_u = x_i^{\mathsf{hnd}}].ind$ for $i = 1, \ldots, j$. If any $D[x_i].type \in \{\mathsf{sks}, \mathsf{ske}\}$, set $l^{\mathsf{hnd}} := \downarrow$.If $l := D[type = \mathsf{list} \wedge arg = (x_1, \ldots, x_j)].ind \neq \downarrow$, then return $l^{\mathsf{hnd}} := \mathsf{ind2hnd}_u(l)$. Otherwise, set $length := \mathsf{list\_len}^*(D[x_1].len, \ldots, D[x_j].len)$ and return $\downarrow$ if $length > \mathsf{max\_len}(k)$. Else set $l^{\mathsf{hnd}} := curhnd_u\mathsf{++}$ and

  $$D \ :\Leftarrow \ (ind := size\mathsf{++}, type := \mathsf{list}, arg := (x_1, \ldots, x_j), hnd_u := l^{\mathsf{hnd}}, len := length).$$

- *i-th projection:* $x^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(l^{\mathsf{hnd}}, i)$, for $1 \leq i \leq \mathsf{max\_len}(k)$.

  If $D[hnd_u = l^{\mathsf{hnd}} \wedge type = \mathsf{list}].arg = (x_1, \ldots, x_j)$ with $j \geq i$, then $x^{\mathsf{hnd}} := \mathsf{ind2hnd}_u(x_i)$, otherwise $x^{\mathsf{hnd}} := \downarrow$.

Further, we used commands to sign a list, to verify a signature, and to retrieve the message from a signature.

- *Signature generation:* $s^{\mathsf{hnd}} \leftarrow \mathsf{sign}(sk^{\mathsf{hnd}}, l^{\mathsf{hnd}})$.

  Let $sk := D[hnd_u = sk^{\mathsf{hnd}} \wedge type = \mathsf{sks}].ind$ and $l := D[hnd_u = l^{\mathsf{hnd}} \wedge type = \mathsf{list}].ind$. If either of these is $\downarrow$ or if $length := \mathsf{sig\_len}^*(k, D[l].len) > \mathsf{max\_len}(k)$, return $\downarrow$. Also return $\downarrow$ if $D[sk].arg[1] \geq \mathsf{max\_skc}(k)$ and $u \neq \mathsf{a}$. Otherwise, set $s^{\mathsf{hnd}} := curhnd_u\mathsf{++}$, $pk := sk + 1$ (recall that key pairs get successive indices), $c := D[sk].arg[1]\mathsf{++}$, and

  $$D :\Leftarrow (ind := size\mathsf{++}, type := \mathsf{sig}, arg := (pk, l, c), hnd_u := s^{\mathsf{hnd}}, len := length).^3$$

- *Signature verification:* $v \leftarrow \mathsf{verify}(s^{\mathsf{hnd}}, pk^{\mathsf{hnd}}, l^{\mathsf{hnd}})$.

  Let $s := D[hnd_u = s^{\mathsf{hnd}} \wedge type = \mathsf{sig}].ind$. If $s = \downarrow$ then return $\downarrow$. Otherwise, let $(pk, l, c) := D[s].arg$. If $D[pk].hnd_u \neq pk^{\mathsf{hnd}}$ or $D[l].hnd_u \neq l^{\mathsf{hnd}}$, then $v := \mathsf{false}$, else $v := \mathsf{true}$.

- *Message retrieval:* $l^{\mathsf{hnd}} \leftarrow \mathsf{msg\_of\_sig}(s^{\mathsf{hnd}})$.

  Let $l := D[hnd_u = s^{\mathsf{hnd}} \wedge type = \mathsf{sig}].arg[2]$ and return $l^{\mathsf{hnd}} := \mathsf{ind2hnd}_u(l)$.

From the set of local adversary commands, which capture additional commands for the adversary at port $\mathsf{in_a}?$, we only describe the commands adv_parse and adv_transform_sig. The first command allows the adversary to retrieve all information that we do not explicitly require to be hidden. This command returns the type and usually all the abstract arguments of a value (with indices replaced by handles), except in the case of ciphertexts. The second command allows the adversary to transform an existing signature that he knows into another one for the same message (which is not excluded by the definition of secure signature schemes).

- *Parameter retrieval:* $(type, arg) \leftarrow \mathsf{adv\_parse}(m^{\mathsf{hnd}})$.

  Let $m := D[hnd_{\mathsf{a}} = m^{\mathsf{hnd}}].ind$ and $type := D[m].type$. In most cases, set $arg := \mathsf{ind2hnd}^*_{\mathsf{a}}(D[m].arg)$. (Recall that this only transforms arguments in $\mathcal{INDS}$.) The only exception is for $type = \mathsf{enc}$, which does not matter in the following.

- *Signature transformation:* $t^{\mathsf{hnd}} \leftarrow \mathsf{adv\_transform\_sig}(s^{\mathsf{hnd}})$.

  Let $s := D[hnd_{\mathsf{a}} = s^{\mathsf{hnd}} \wedge type = \mathsf{sig}].ind$. If $s = \downarrow$ then return $\downarrow$. Otherwise let $(pk, l, c) := D[s].arg$. Set $t^{\mathsf{hnd}} := curhnd_{\mathsf{a}}\mathsf{++}$ and

  $$D \quad :\Leftarrow \quad (ind := size\mathsf{++}, type := \mathsf{sig}, arg := (pk, l, \mathsf{false}), hnd_{\mathsf{a}} := t^{\mathsf{hnd}}, len := D[s].len).$$

We finally describe the commands for sending messages on insecure channels. In the second one, the adversary sends list l to v, pretending to be u.

- $\mathsf{send\_i}(v, l^{\mathsf{hnd}})$, for $v \in \{1, \ldots, n\}$ at port $\mathsf{in}_u?$ for $u \in \mathcal{H}$.

  Let $l^{\mathsf{ind}} := D[hnd_u = l^{\mathsf{hnd}} \wedge type = \mathsf{list}].ind$. If $l^{\mathsf{ind}} \neq \downarrow$, output $(u, v, \mathsf{i}, \mathsf{ind2hnd}_{\mathsf{a}}(l^{\mathsf{ind}}))$ at $\mathsf{out_a}!$.

- $\mathsf{adv\_send\_i}(u, v, l^{\mathsf{hnd}})$, for $u \in \{1, \ldots, n\}$ and $v \in \mathcal{H}$ at port $\mathsf{in_a}?$.

  Let $l^{\mathsf{ind}} := D[hnd_{\mathsf{a}} = l^{\mathsf{hnd}} \wedge type = \mathsf{list}].ind$. If $l^{\mathsf{ind}} \neq \downarrow$, output $(u, v, \mathsf{i}, \mathsf{ind2hnd}_v(l^{\mathsf{ind}}))$ at $\mathsf{out}_v!$.

[3] This type also exists with $c = \mathsf{false}$ due to the command adv_transform_sig.

B Postponed Proofs

B.1 Proof of the Invariants

The statements *no modification* and *unique payment entries* can easily be verified by inspection of the algorithms.

Proof. (Correct Signing (Client Part)) We refer to Step i of Algorithm j as Step $j.i$. Let $u \in \mathcal{M}^{\mathsf{client}} \cap \mathcal{H}$, $i \in \mathcal{INDS}$, $t_2 \in \mathbb{N}$, and r a trace arising in runs of $Sys^{\mathsf{PS,id}}$. Let $l := r_{t_2} : D[i].arg[2]$, $y_j := r_{t_2} : D[l].arg[j]$, and $x_j := r_{t_2} : D[y_j].arg[1]$ for $j = 1, \ldots, 5$. The proof is performed by induction over t_2. Assume that the entry $D[i]$ is generated at time t_2 in trace r and assume further that the invariant holds at all times prior than t_2. The only commands of the ideal cryptographic library that generate an entry $D[i]$ with $D[i].type = \mathsf{sig}$ are $\mathsf{sign}(sk^{\mathsf{hnd}}, m^{\mathsf{hnd}})$ and $\mathsf{adv\_transform\_sig}(s^{\mathsf{hnd}})$.

We first consider the command $\mathsf{adv\_transform\_sig}(s^{\mathsf{hnd}})$, which can only be input at port $\mathsf{in_a}?$. Let $s := r_{t_2} : D[hnd_{\mathsf{a}} = s^{\mathsf{hnd}} \wedge type = \mathsf{sig}].ind$. If $D[i]$ is generated in this transition, the definition of the command implies $s \neq \downarrow$, $D[s].arg[1] = D[i].arg[1]$, and $D[s].arg[2] = l$. Hence we can use our induction hypothesis with $D[s]$ instead of $D[i]$ which immediately finishes the proof of this case, since $D[s]$ and $D[i]$ have identical first and second arguments.

Now consider a command $\mathsf{sign}(sk^{\mathsf{hnd}}, m^{\mathsf{hnd}})$ input by w at w's local port. (This can be either $\mathsf{PS\_in}_w?$ if $w \in \mathcal{H}$ or $\mathsf{in_a}?$ if $w = \mathsf{a}$.) Let $sk := r_{t_2} : D[hnd_w = sk^{\mathsf{hnd}} \wedge type = \mathsf{sks}].ind$ and $m := r_{t_2} : D[hnd_w = m^{\mathsf{hnd}} \wedge type = \mathsf{list}].ind$. If $D[i]$ is generated in this transition, the definition of the sign command implies $sk \neq \downarrow$ and $D[i].arg[1,2] = (pk, m)$ where $pk = sk + 1$ (recall that secret and public keys get successive indices). This yields $sk = D[i].arg[1] - 1 = pks_u - 1 = sks_u$. Furthermore, $sk \neq \downarrow$ implies $sk^{\mathsf{hnd}} \neq \downarrow$. Since we initially have $D[sks_u].hnd_w \neq \downarrow$ only if $z = u$, cf. Section 6.1, and since entries of type sks cannot be sent to other parties by definition, we conclude that $w = u$. Hence the client u must have input the sign command at port $\mathsf{PS\_in}_u?$. Inspection of Algorithm 1, 2, and 3 shows that sign commands for u exist in Steps 1.10, 2.14, and 3.4.

If $D[i]$ is generated in Step 1.10, then Steps 1.4-1.9, the definition of the command store, and Convention 1 imply $x_4 = u$ and $D[y_j].type = data$ for $j = 1, \ldots, 5$. Furthermore, Algorithm 1 is started by an input (pay, d, p, v) at $\mathsf{PS\_in}_u?$ at some time t_1, and Steps 1.4-1.9 imply $d = x_2$, $p = x_3$, and $v = x_5$.

If $D[i]$ is generated in Step 2.14, then Steps 2.7, 2.12, and 2.13, the definition of the commands store and retrieve, and Convention 1 imply $D[y_j].type = data$ for $j = 1, \ldots, 5$. The look-up in Step 2.8 and the condition in Step 2.9 further yield $x_4 = u$, $x_5 = v$, and $ind := D_u^{\mathsf{PS}}[desc = x_2 \wedge price = x_3 \wedge merch = x_5 \wedge status = \mathsf{pay}].ind \neq \downarrow$. By *no modification*, the entry $D_u^{\mathsf{PS}}[ind]$ must have been created in Step 1.14. Algorithm 1 is started by an input (pay, d, p, v) at $\mathsf{PS\_in}_u?$ at some time t_1, with $D_u^{\mathsf{PS}}[ind].desc = d$, $D_u^{\mathsf{PS}}[ind].price = p$, and $D_u^{\mathsf{PS}}[ind].merch = x_5$. This yields $d = x_2$, $p = x_3$, and $v = x_5$.

If $D[i]$ is generated in Step 3.4, Steps 3.2 and 3.3 immediately yield $x_1 = \mathsf{dispute}$, hence nothing needs to be proved. ∎

Proof. (Correct Storing (Client Part)) We use the notation of the theorem. Let further $i^w := r_t : D[hnd_u = s^w].ind$ for $w \in \{\mathsf{m}, \mathsf{ac}\}$. Assume that $\mathsf{PS\_out}_u!(\mathsf{paid}, d, p, v) \in r_t$. This output may only occur in Step 2.28. In the following, we use the notation of Algorithm 2, hence we have $(x_2, x_3, x_5) = (d, p, v)$. Step 2.25 and 2.26 together with *unique payment entries* imply $i = i^{\mathsf{m}} = i^{\mathsf{ac}}$. We first show the statements about the signature of the acquirer.

Because of $i = i^{\mathsf{ac}}$ Step 2.27 implies $s^{\mathsf{ac}} = l_2^{\mathsf{hnd}}$. Thus $pk^{\mathsf{ac}} = r_t : D[hnd_u = l_2^{\mathsf{hnd}}].arg[1]$ and $l^{\mathsf{ac}} = r_t : D[hnd_u = l_2^{\mathsf{hnd}}].arg[2]$. Now Step 2.22 and 2.26 the definition of the command verify immediately imply $r_t : D[hnd_u = s^{\mathsf{ac}}].type = r_t : D[hnd_u = l_2^{\mathsf{hnd}}].type = \mathsf{sig}$ and $r_t : D[pk'].hnd_u = pks_{u,\mathsf{ac}}^{\mathsf{hnd}}$, where $pk' = r_t : D[hnd_u = l_2^{\mathsf{hnd}}].arg[1] = r_t : D[hnd_u = s^{\mathsf{ac}}].arg[1] = pk^{\mathsf{ac}}$. Hence $r_t : D[pk^{\mathsf{ac}}].hnd_u = pks_{u,\mathsf{ac}}^{\mathsf{hnd}}$. Moreover, we obtain $x_j^{\mathsf{ac}} = x_j$ by Step 2.23, 2.24, and the definition of list_proj and retrieve, hence

Algorithm 4 Merchant: Evaluation of User Inputs for Receiving in $\mathsf{M}_u^{\mathsf{PS}}$

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

1: **if** $D_u^{\mathsf{PS}}[desc = d \wedge price = p \wedge client = v] = \downarrow$ **then**
2: $D_u^{\mathsf{PS}} :\Leftarrow (cur\_ind_u\mathsf{++}, d, p, v, \downarrow, \downarrow)$.
3: $\mathsf{invoice}^{\mathsf{hnd}} \leftarrow \mathsf{store}(\mathsf{invoice})$.
4: $d^{\mathsf{hnd}} \leftarrow \mathsf{store}(d)$.
5: $p^{\mathsf{hnd}} \leftarrow \mathsf{store}(p)$.
6: $u^{\mathsf{hnd}} \leftarrow \mathsf{store}(u)$.
7: $v^{\mathsf{hnd}} \leftarrow \mathsf{store}(v)$.
8: $l^{\mathsf{hnd}} \leftarrow \mathsf{list}(\mathsf{invoice}^{\mathsf{hnd}}, d^{\mathsf{hnd}}, p^{\mathsf{hnd}}, v^{\mathsf{hnd}}, u^{\mathsf{hnd}})$.
9: $s^{\mathsf{hnd}} \leftarrow \mathsf{sign}(l^{\mathsf{hnd}})$.
10: $m^{\mathsf{hnd}} \leftarrow \mathsf{list}(\mathsf{invoice}^{\mathsf{hnd}}, s^{\mathsf{hnd}})$.
11: $\mathsf{send\_i}(v, m^{\mathsf{hnd}})$.
12: **end if**

$x_2^{\mathsf{ac}} = d$, $x_3^{\mathsf{ac}} = p$, and $x_5^{\mathsf{ac}} = v$. Step 2.26 additionally yields $x_1^{\mathsf{ac}} = \mathsf{auth\_response}$, which finishes this part.

To show the statements about the signature of the merchant, we exploit that Step 2.25 implies $r_t : D[i].status = \mathsf{processed}$. The status of $D[i]$ may only have been set to processed in Step 2.11 and 1.3. In the first case, 2.10 yields $s^{\mathsf{m}} = l_2^{\mathsf{hnd}}$. The statement for this case is then proved analogously to the statement about the acquirer's signature with Steps 2.22-2.27 replaced by Steps 2.5-2.10. In the second case, Step 1.2 implies that $D[i].status = \mathsf{invoice}$ somewhen before time t, hence this entry must have been created in Step 2.18. Again the proof works identically as for the acquirer, with Steps 2.22-2.25 replaced by Steps 2.5-2.8, and Step 2.26 replaced by Step 2.17.

■

B.2 Proof of the Overall Integrity Property

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

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

Assume $\mathsf{PS\_out}_u!(\mathsf{paid}, d, p, v) \in r_{t_2}$. This output may only occur in Step 2.28. Step 2.25 and 2.26 together with *unique payment entries* and *no modification* implies $r_{t_2} : D_u^{\mathsf{PS}}[i].status = \mathsf{processed}$, and the status of $D_u^{\mathsf{PS}}[i]$ 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 (pay, d, p, v) at port $\mathsf{PS\_in}_u?$ at some time $t_1 < t_2$. In the second case, Step 2.9 ensures $r_{t_1'} : D_u^{\mathsf{PS}}[i].status = \mathsf{pay}$ for some $t_1' < t_2$, hence this entry was created in Step 1.14. Again the algorithm was activated on inputs (pay, d, p, v) at port $\mathsf{PS\_in}_u?$ at some time $t_1 < t_1' < t_2$.

Assume $\mathsf{PS\_out}_v!(\mathsf{received}, d, p, u) \in r_{t_2}$ and $v \in \mathcal{H}$. This output may only occur in Step 6.28. Let $j := r_{t_2} : D_v^{\mathsf{PS}}[desc = d \wedge price = p \wedge client = u].ind$. Step 6.23 and 6.24 together with *unique payment*

Algorithm 5 Acquirer: Evaluation of User Inputs for Allow in $\mathsf{M}_{\mathsf{ac}}^{\mathsf{PS}}$

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

1: $i := D_{\mathsf{ac}}^{\mathsf{PS}}[desc = d \wedge price = p \wedge client = u \wedge merch = v].ind.$
2: **if** $i \neq \downarrow \wedge D_{\mathsf{ac}}^{\mathsf{PS}}[i].status = \mathsf{auth\_request}$ **then**
3: $\quad D_{\mathsf{ac}}^{\mathsf{PS}}[i].status := \mathsf{processed}.$
4: $\quad \mathsf{auth\_response}^{\mathsf{hnd}} \leftarrow \mathsf{store}(\mathsf{auth\_response}).$
5: $\quad d^{\mathsf{hnd}} \leftarrow \mathsf{store}(d).$
6: $\quad p^{\mathsf{hnd}} \leftarrow \mathsf{store}(p).$
7: $\quad u^{\mathsf{hnd}} \leftarrow \mathsf{store}(u).$
8: $\quad v^{\mathsf{hnd}} \leftarrow \mathsf{store}(v).$
9: $\quad l^{\mathsf{hnd}} \leftarrow \mathsf{list}(\mathsf{auth\_response}^{\mathsf{hnd}}, d^{\mathsf{hnd}}, p^{\mathsf{hnd}}, u^{\mathsf{hnd}}, v^{\mathsf{hnd}}).$
10: $\quad s^{\mathsf{hnd}} \leftarrow \mathsf{sign}(sks_{\mathsf{ac}}^{\mathsf{hnd}}, l^{\mathsf{hnd}}).$
11: $\quad m^{\mathsf{hnd}} \leftarrow \mathsf{list}(\mathsf{auth\_response}^{\mathsf{hnd}}, s^{\mathsf{hnd}}).$
12: $\quad \mathsf{send\_i}(v, m^{\mathsf{hnd}}).$
13: **else if** $i = \downarrow$ **then**
14: $\quad D_{\mathsf{ac}}^{\mathsf{PS}} :\Leftarrow (cur\_ind_{\mathsf{ac}}\mathsf{++}, d, p, u, v, \downarrow, \downarrow, \mathsf{allow}).$
15: **end if**

entries and *no modification* imply $r_{t_2} : D_v^{\mathsf{PS}}[j].sig_{\mathsf{c}} \neq \downarrow$. The only step where v assigns a value different from $\downarrow$ to attribute sig_{c} is in Step 6.10. Let $l_2^{\mathsf{hnd}} = r_{t_2} : D_v^{\mathsf{PS}}[j].sig_{\mathsf{c}}$ and let $i := r_{t_2} : D[hnd_v = l_2^{\mathsf{hnd}}].ind$. Let $l := r_{t_2} : D[i].arg[2]$, $y_j := r_{t_2} : D[l].arg[j]$, and $x_j := r_{t_2} : D[y_j].arg[1]$ for $j = 1, \ldots, 5$. Then Steps 6.4-6.9, the definition of the commands verify, list_proj, and retrieve imply $r_{t_2} : D[i].type = \mathsf{sig}$, $r_{t_2} : D[i].arg[1] = pks_u$, $x_1 = \mathsf{payment}$, $x_2 = d$, $x_3 = p$, 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 $\mathsf{PS\_in}_u?(\mathsf{pay}, d, p, v) \in r_{t_1}$.

Assume $\mathsf{PS\_out}_{\mathsf{ac}}!(\mathsf{transfer}, d, p, u, v) \in tr_{t_2}$ and $\mathsf{ac} \in \mathcal{H}$. 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 B.2 *For the payment system from Section 3.3 and the correct disputing property* $Req^{\mathsf{corr\_disp}}$*, we have* $Sys^{\mathsf{PS,id}} \models^{\mathsf{perf}} Req^{\mathsf{corr\_disp}}$.

Proof. Let $d \in \Sigma^*$, $p \in \mathbb{N}$, $u \in \mathcal{M}^{\mathsf{client}} \cap \mathcal{H}$, $v \in \mathcal{M}^{\mathsf{merchant}}$, $t_3 \in \mathbb{N}$, and r a trace arising in runs of $Sys^{\mathsf{PS,id}}$. Again, we only show the client part of the statement, i.e., assume $\mathsf{PS\_out}_{\mathsf{ttp}}!(\mathsf{dispute}, \mathsf{paid}, \mathsf{false}, d, p, u, v) \in r_{t_3}$. The output must have occured in Step 10.20.

The algorithm is invoked only on input $(w, \mathsf{ttp}, \mathsf{i}, l^{\mathsf{hnd}})$. Let $i := D[hnd_{\mathsf{ttp}} = l^{\mathsf{hnd}}].arg[1]$, $l_1 := D[i].arg[2]$, $(pk, s_1, s_2) := D[l_1].arg[1, 2, 3]$, $y_j := D[s_1].arg[j]$, and $x_j := D[y_j].arg[1]$ 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 $D[i].type = \mathsf{sig}$ and $pk = pks_u$. It can then be shown along the lines of the proof of *correct signing* that $\mathsf{M}_u^{\mathsf{PS}}$ must have input a command $\mathsf{sign}(\mathsf{dispute}, D[s_1].hnd_u, D[s_2].hnd_u)$ such that $x_1 = \mathsf{payment}, x_2 = d, x_3 = p, x_4 = u$, and $x_5 = v$. The only syntactically matching sign command is in Step 3.4, and is executed only on input $(\mathsf{dispute}, d, p, v)$ at $\mathsf{PS\_in}_u?$ at some time $t_2 < t_3$. We remain to show the nonexistence of an output (paid, d, p, v) at $\mathsf{PS\_out}_u!$ for all times $t_1 < t_2$. We prove this by contradiction. Assume that there exists $t_1 \in \mathbb{N}$ such that $\mathsf{PS\_out}_u!(\mathsf{paid}, d, p, v) \in r_{t_1}$. Then by *correct storing* both signatures sig_{m} and 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 B.3 *For the payment system from Section 3.3 and the no framing property* $Req^{\mathsf{no\_frame}}$*, we have* $Sys^{\mathsf{PS,id}} \models^{\mathsf{perf}} Req^{\mathsf{no\_frame}}$.

Proof. Let $x \in \{\mathsf{received}, \mathsf{transfer}\}$, $d \in \Sigma^*$, $p \in \mathbb{N}$, $u \in \mathcal{H}^{\mathsf{client}}$, $v \in \mathcal{M}^{\mathsf{merchant}}$, $t_2 \in \mathbb{N}$, and r be a trace arising in runs of $Sys^{\mathsf{PS,id}}$. Again we prove only the client part of the statement, i.e., we prove the statement $\mathsf{PS\_out}_{\mathsf{ttp}}!(\mathsf{dispute}, d, p, u, v) \in r_{t_2} \Rightarrow \exists t_1 < t_2 : \mathsf{PS\_in}_u?(\mathsf{pay}, d, p, v) \in r_{t_1}$.

Assume $\mathsf{PS\_out}_{\mathsf{ttp}}?(\mathsf{dispute}, x, \mathsf{true}, d, p, u, v) \in r_{t_2}$ for $x = \mathsf{received}$ ($x = \mathsf{transfer}$). This output occurs only in Step 10.26 (in Step 10.34). With l_2^{hnd} as in Algorithm 10, let $i := r_{t_2} : D[hnd_{\mathsf{ttp}} = l_2^{\mathsf{hnd}}].ind$. Now Step 10.23 and 10.25 (Step 10.31 and 10.33) ensure that $D[i].type = \mathsf{sig}$, $D[i].arg[1] = pks_{x_4} = D[i].arg[1] = pks_u$, and $x_1 = \mathsf{payment}$. This implies that $D[i]$ meets the prerequisites *correct signing*, hence there exists $t_1 < t_2$ such that $\mathsf{PS\_in}_u?(\mathsf{pay}, d, p, v) \in r_{t_1}$. ∎

Algorithm 6 Merchant: Evaluation of Inputs from the Cryptographic Library in $\mathsf{M}_u^{\mathsf{PS}}$

Input: $(v, u, \mathsf{i}, l^{\mathsf{hnd}})$ at out_u? with $v \in \mathcal{M}^{\mathsf{client}} \cup \{\mathsf{ac}\}$.

1: $l_j^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(l^{\mathsf{hnd}}, j)$ for $j = 1, 2$.
2: $l_1 \leftarrow \mathsf{retrieve}(l_1^{\mathsf{hnd}})$.
3: **if** $l_1 = \mathsf{payment} \wedge v \neq \mathsf{ac}$ **then**
4: $m_2^{\mathsf{hnd}} \leftarrow \mathsf{msg\_of\_sig}(l_2^{\mathsf{hnd}})$.
5: $b \leftarrow \mathsf{verify}(l_2^{\mathsf{hnd}}, pks_{u,v}^{\mathsf{hnd}}, m_2^{\mathsf{hnd}})$
6: $x_j^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(m_2^{\mathsf{hnd}}, j)$ for $j = 1, \ldots, 5$.
7: $x_j \leftarrow \mathsf{retrieve}(x_j^{\mathsf{hnd}})$ for $j = 1, \ldots, 5$.
8: $i := D_u^{\mathsf{PS}}[desc = x_2 \wedge price = x_3 \wedge client = x_4 \wedge sig_{\mathsf{c}} = \downarrow].ind$.
9: **if** $x_1 = \mathsf{payment} \wedge x_4 = v \wedge x_5 = u \wedge b = \mathsf{true} \wedge i \neq \downarrow$ **then**
10: $D_u^{\mathsf{PS}}[i].sig_{\mathsf{c}} := l_2^{\mathsf{hnd}}$.
11: $\mathsf{invoice}^{\mathsf{hnd}} \leftarrow \mathsf{store}(\mathsf{invoice})$.
12: $m_1^{\mathsf{hnd}} \leftarrow \mathsf{list}(\mathsf{invoice}^{\mathsf{hnd}}, x_2^{\mathsf{hnd}}, x_3^{\mathsf{hnd}}, x_4^{\mathsf{hnd}}, x_5^{\mathsf{hnd}})$.
13: $s_1^{\mathsf{hnd}} \leftarrow \mathsf{sign}(sks_u^{\mathsf{hnd}}, m_1^{\mathsf{hnd}})$.
14: $\mathsf{auth\_request}^{\mathsf{hnd}} \leftarrow \mathsf{store}(\mathsf{auth\_request})$.
15: $m^{\mathsf{hnd}} \leftarrow \mathsf{list}(\mathsf{auth\_request}^{\mathsf{hnd}}, l_2^{\mathsf{hnd}}, s_1^{\mathsf{hnd}})$.
16: $\mathsf{send\_i}(\mathsf{ac}, m^{\mathsf{hnd}})$
17: **end if**
18: **else if** $x_1 = \mathsf{auth\_response} \wedge v = \mathsf{ac}$ **then**
19: $m_2^{\mathsf{hnd}} \leftarrow \mathsf{msg\_of\_sig}(l_2^{\mathsf{hnd}})$.
20: $b \leftarrow \mathsf{verify}(l_2^{\mathsf{hnd}}, pks_{u,v}^{\mathsf{hnd}}, m_2^{\mathsf{hnd}})$
21: $x_j^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(m_2^{\mathsf{hnd}}, j)$ for $j = 1, \ldots, 5$.
22: $x_j \leftarrow \mathsf{retrieve}(x_j^{\mathsf{hnd}})$ for $j = 1, \ldots, 5$.
23: $i := D_u^{\mathsf{PS}}[desc = x_2 \wedge price = x_3 \wedge client = x_4 \wedge sig_{\mathsf{c}} \neq \downarrow].ind$.
24: **if** $x_5 = u \wedge b = \mathsf{true} \wedge i \neq \downarrow$ **then**
25: $D_u^{\mathsf{PS}}[i].sig_{\mathsf{ac}} := l_2^{\mathsf{hnd}}$.
26: $\mathsf{confirm}^{\mathsf{hnd}} \leftarrow \mathsf{store}(\mathsf{confirm})$.
27: $m^{\mathsf{hnd}} \leftarrow \mathsf{list}(\mathsf{confirm}^{\mathsf{hnd}}, l_2^{\mathsf{hnd}})$.
28: Output $(\mathsf{received}, x_2, x_3, x_4)$ at $\mathsf{PS\_out}_u$!.
29: **end if**
30: **end if**

Algorithm 7 Merchant: Evaluation of User Inputs for Disputes in $\mathsf{M}_u^{\mathsf{PS}}$

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

1: **if** $i := D_u^{\mathsf{PS}}[desc = d \wedge price = p \wedge client = v \wedge sig_{\mathsf{c}} \neq \downarrow \wedge sig_{\mathsf{ac}} \neq \downarrow].ind \neq \downarrow$ **then**
2: $\mathsf{dispute}^{\mathsf{hnd}} \leftarrow \mathsf{store}(\mathsf{dispute})$.
3: $l^{\mathsf{hnd}} \leftarrow \mathsf{list}(\mathsf{dispute}^{\mathsf{hnd}}, D_u^{\mathsf{PS}}[i].sig_{\mathsf{c}}, D_u^{\mathsf{PS}}[i].sig_{\mathsf{ac}})$.
4: $s^{\mathsf{hnd}} \leftarrow \mathsf{sign}(sks_u^{\mathsf{hnd}}, l^{\mathsf{hnd}})$.
5: $m^{\mathsf{hnd}} \leftarrow \mathsf{list}(s^{\mathsf{hnd}})$.
6: $\mathsf{send\_i}(\mathsf{ttp}, m^{\mathsf{hnd}})$.
7: **end if**

Algorithm 8 Acquirer: Evaluation of Inputs from the Cryptographic Library in $\mathsf{M}_{\mathsf{ac}}^{\mathsf{PS}}$

Input: $(v, \mathsf{ac}, \mathsf{i}, l^{\mathsf{hnd}})$ at $\mathsf{out}_{\mathsf{ac}}$? with $v \in \mathcal{M}^{\mathsf{merchant}}$.

$l_j^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(l^{\mathsf{hnd}}, j)$ for $j = 1, 2, 3$.
$l_1 \leftarrow \mathsf{retrieve}(l_1^{\mathsf{hnd}})$.
if $l_1 \neq \mathsf{auth\_request}$ **then**
  Abort
end if
$m_j^{\mathsf{hnd}} \leftarrow \mathsf{msg\_of\_sig}(l_j^{\mathsf{hnd}})$ for $j = 2, 3$.
$x_j^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(m_2^{\mathsf{hnd}}, j)$ for $j = 1, \ldots, 5$.
$x_j \leftarrow \mathsf{retrieve}(x_j^{\mathsf{hnd}})$ for $j = 1, \ldots, 5$.
$y_j^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(m_3^{\mathsf{hnd}}, j)$ for $j = 1, \ldots, 5$.
$y_j \leftarrow \mathsf{retrieve}(y_j^{\mathsf{hnd}})$ for $j = 1, \ldots, 5$.
$b_2 \leftarrow \mathsf{verify}(l_2^{\mathsf{hnd}}, pks_{\mathsf{ac},x_4}^{\mathsf{hnd}}, m_2^{\mathsf{hnd}})$.
$b_3 \leftarrow \mathsf{verify}(l_3^{\mathsf{hnd}}, pks_{\mathsf{ac},v}^{\mathsf{hnd}}, m_3^{\mathsf{hnd}})$.
if $x_1 = \mathsf{payment} \wedge y_1 = \mathsf{invoice} \wedge b_2 = b_3 = \mathsf{true} \wedge x_5 = v \wedge \forall j = 2, \ldots, 5\colon x_j = y_j$ **then**
  $i := D_{\mathsf{ac}}^{\mathsf{PS}}[desc = x_2 \wedge price = x_3 \wedge client = x_4 \wedge merch = x_5]$.
  if $i \neq \downarrow \wedge D_{\mathsf{ac}}^{\mathsf{PS}}[i].status = \mathsf{allow}$ **then**
    $D_{\mathsf{ac}}^{\mathsf{PS}}[i].sig_{\mathsf{c}} := l_2^{\mathsf{hnd}}$.
    $D_{\mathsf{ac}}^{\mathsf{PS}}[i].sig_{\mathsf{m}} := l_3^{\mathsf{hnd}}$.
    $D_{\mathsf{ac}}^{\mathsf{PS}}[i].status := \mathsf{processed}$.
    $\mathsf{auth\_response}^{\mathsf{hnd}} \leftarrow \mathsf{store}(\mathsf{auth\_response})$.
    $m_1^{\mathsf{hnd}} \leftarrow \mathsf{list}(\mathsf{auth\_response}^{\mathsf{hnd}}, x_2^{\mathsf{hnd}}, x_3^{\mathsf{hnd}}, x_4^{\mathsf{hnd}}, x_5^{\mathsf{hnd}})$.
    $s^{\mathsf{hnd}} \leftarrow \mathsf{sign}(sks_{\mathsf{ac}}^{\mathsf{hnd}}, m_1^{\mathsf{hnd}})$.
    $m^{\mathsf{hnd}} \leftarrow \mathsf{list}(\mathsf{auth\_response}^{\mathsf{hnd}}, s^{\mathsf{hnd}})$.
    Output $(\mathsf{transfer}, x_2, x_3, x_4, x_5)$ at $\mathsf{PS\_out}_{\mathsf{ac}}$!.
    $\mathsf{send\_i}(v, m^{\mathsf{hnd}})$.
  else if $i = \downarrow$ **then**
    $D_{\mathsf{ac}}^{\mathsf{PS}} :\Leftarrow (cur\_ind_{\mathsf{ac}}\mathsf{++}, x_2, x_3, x_4, x_5, l_1^{\mathsf{hnd}}, l_2^{\mathsf{hnd}}, \mathsf{auth\_request})$.
  end if
end if

Algorithm 9 Acquirer: Evaluation of User Inputs for Disputes in $\mathsf{M}_{\mathsf{ac}}^{\mathsf{PS}}$

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

if $i := D_{\mathsf{ac}}^{\mathsf{PS}}[desc = d \wedge price = p \wedge client = u \wedge merch = v \wedge sig_{\mathsf{c}} \neq \downarrow \wedge sig_{\mathsf{m}} \neq \downarrow].ind \neq \downarrow$ **then**
  $\mathsf{dispute}^{\mathsf{hnd}} \leftarrow \mathsf{store}(\mathsf{dispute})$.
  $l^{\mathsf{hnd}} \leftarrow \mathsf{list}(\mathsf{dispute}^{\mathsf{hnd}}, D_{\mathsf{ac}}^{\mathsf{PS}}[i].sig_{\mathsf{c}}, D_{\mathsf{ac}}^{\mathsf{PS}}[i].sig_{\mathsf{m}})$.
  $s^{\mathsf{hnd}} \leftarrow \mathsf{sign}(sks_{\mathsf{ac}}^{\mathsf{hnd}}, l^{\mathsf{hnd}})$.
  $m^{\mathsf{hnd}} \leftarrow \mathsf{list}(s^{\mathsf{hnd}})$.
  $\mathsf{send\_i}(\mathsf{ttp}, m^{\mathsf{hnd}})$.
end if

Algorithm 10 TTP: Evaluation of Inputs from the Cryptographic Library in $\mathsf{M}^{\mathsf{PS}}_{\mathsf{ttp}}$

Input: $(v, \mathsf{ttp}, \mathsf{i}, l^{\mathsf{hnd}})$ at $\mathsf{out}_{\mathsf{ttp}}$? for $v \in \mathcal{M} \setminus \{\mathsf{ttp}\}$.

$s^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(l^{\mathsf{hnd}}, 1)$.
$l^{*^{\mathsf{hnd}}} \leftarrow \mathsf{msg\_of\_sig}(s^{\mathsf{hnd}})$
$b_1 \leftarrow \mathsf{verify}(s^{\mathsf{hnd}}, pks^{\mathsf{hnd}}_{\mathsf{ttp},v}, l^{*^{\mathsf{hnd}}})$.
$l_j^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(l^{*^{\mathsf{hnd}}}, j)$ for $j = 1, 2, 3$.
$l_1 \leftarrow \mathsf{retrieve}(l_1^{\mathsf{hnd}})$.
if $l_1 \neq \mathsf{dispute} \vee b_1 \neq \mathsf{true}$ **then**
  Abort
end if
$m_j^{\mathsf{hnd}} \leftarrow \mathsf{msg\_of\_sig}(l_j^{\mathsf{hnd}})$ for $j = 2, 3$.
$x_j^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(m_2^{\mathsf{hnd}}, j)$ for $j = 1, \ldots, 5$.
$x_j \leftarrow \mathsf{retrieve}(x_j^{\mathsf{hnd}})$ for $j = 1, \ldots, 5$.
$y_j^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(m_3^{\mathsf{hnd}}, j)$ for $j = 1, \ldots, 5$.
$y_j \leftarrow \mathsf{retrieve}(y_j^{\mathsf{hnd}})$ for $j = 1, \ldots, 5$.
if $v \in \mathcal{M}^{\mathsf{client}}$ **then**
  $b_2 \leftarrow \mathsf{verify}(l_2^{\mathsf{hnd}}, pks^{\mathsf{hnd}}_{\mathsf{ttp},x_5}, m_2^{\mathsf{hnd}})$.
  $b_3 \leftarrow \mathsf{verify}(l_3^{\mathsf{hnd}}, pks^{\mathsf{hnd}}_{\mathsf{ttp},\mathsf{ac}}, m_3^{\mathsf{hnd}})$.
  if $x_1 = \mathsf{invoice} \wedge y_1 = \mathsf{auth\_response} \wedge x_4 = v \wedge b_2 = b_3 = \mathsf{true} \wedge \forall j = 2, \ldots, 5\colon x_j = y_j$ **then**
    Output $(\mathsf{dispute}, \mathsf{paid}, \mathsf{true}, x_2, x_3, x_4, x_5)$ at $\mathsf{PS\_out}_{\mathsf{ttp}}$!.
  else
    Output $(\mathsf{dispute}, \mathsf{paid}, \mathsf{false}, x_2, x_3, x_4, x_5)$ at $\mathsf{PS\_out}_{\mathsf{ttp}}$!.
  end if
else if $v \in \mathcal{M}^{\mathsf{merchant}}$ **then**
  $b_2 \leftarrow \mathsf{verify}(l_2^{\mathsf{hnd}}, pks^{\mathsf{hnd}}_{\mathsf{ttp},x_4}, m_2^{\mathsf{hnd}})$.
  $b_3 \leftarrow \mathsf{verify}(l_3^{\mathsf{hnd}}, pks^{\mathsf{hnd}}_{\mathsf{ttp},\mathsf{ac}}, m_3^{\mathsf{hnd}})$.
  if $x_1 = \mathsf{payment} \wedge y_1 = \mathsf{auth\_response} \wedge x_5 = v \wedge b_2 = b_3 = \mathsf{true} \wedge \forall j = 2, \ldots, 5\colon x_j = y_j$ **then**
    Output $(\mathsf{dispute}, \mathsf{received}, \mathsf{true}, x_2, x_3, x_4, x_5)$ at $\mathsf{PS\_out}_{\mathsf{ttp}}$!.
  else
    Output $(\mathsf{dispute}, \mathsf{received}, \mathsf{false}, x_2, x_3, x_4, x_5)$ at $\mathsf{PS\_out}_{\mathsf{ttp}}$!.
  end if
else if $v = \mathsf{ac}$ **then**
  $b_2 \leftarrow \mathsf{verify}(l_2^{\mathsf{hnd}}, pks^{\mathsf{hnd}}_{\mathsf{ttp},x_4}, m_2^{\mathsf{hnd}})$.
  $b_3 \leftarrow \mathsf{verify}(l_3^{\mathsf{hnd}}, pks^{\mathsf{hnd}}_{\mathsf{ttp},x_5}, m_3^{\mathsf{hnd}})$.
  if $x_1 = \mathsf{payment} \wedge y_1 = \mathsf{invoice} \wedge b_2 = b_3 = \mathsf{true} \wedge \forall j = 2, \ldots, 5\colon x_j = y_j$ **then**
    Output $(\mathsf{dispute}, \mathsf{transfer}, \mathsf{true}, x_2, x_3, x_4, x_5)$ at $\mathsf{PS\_out}_{\mathsf{ttp}}$!.
  else
    Output $(\mathsf{dispute}, \mathsf{transfer}, \mathsf{false}, x_2, x_3, x_4, x_5)$ at $\mathsf{PS\_out}_{\mathsf{ttp}}$!.
  end if
end if