Practical, Anonymous, and Publicly Linkable Universally-Composable Reputation Systems

Johannes Blömer[*] Fabian Eidens[*] Jakob Juhnke[*]

January 7, 2018
Department of Computer Science
Paderborn University, Germany
{bloemer, juhnke, feidens}@mail.uni-paderborn.de

Abstract

We consider reputation systems in the Universal Composability Framework where users can anonymously rate each others products that they purchased previously. To obtain trustworthy, reliable, and honest ratings, users are allowed to rate products only once. Everybody is able to detect users that rate products multiple times. In this paper we present an ideal functionality for such reputation systems and give an efficient realization that is usable in practical applications.

Keywords: Reputation, Trust, Anonymity, Universal Composability

1 Introduction

Reputation systems provide valuable information about previous transactions and are popular tools to measure trustworthiness of interacting parties. This measurement relies on the existence of a large number of ratings for one specific subject. But in most practical applications the process of rating reveals, besides the actual rating, many information about the rater. Providers of reputation systems use this information in many different ways, e.g. for profiling users, which are not necessarily desired by the users. Moreover, users can feel compelled to rate "dishonestly/benevolent" when they fear negative consequences from negative ratings. Therefore, it is important that the process of rating does not reveal more information than the actual rating. Besides that, reputation systems need to be protected against various attacks to provide trustworthy, reliable and honest ratings. These attacks include self-rating attacks (also known as self-promoting attacks), Sybil attacks, whitewashing attacks, bad mouthing attacks, ballot stuffing attacks, and value imbalance attacks. Both the privacy concerns and the prevention of attacks are

[*]This author was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre On-The-Fly Computing (SFB 901).

discussed frequently in the literature, e.g. [ACBM08, BPS$^+$17, CSK13, Del00, HBBS13, HZNR09, Ker09, PLS14, Ste06, ZWC$^+$16], albeit they are not considered simultaneously.

Further important security properties for reputation systems are *anonymity*, *(public) linkability*, *traceability*, and *non-frameability*, as discussed in [ACBM08, BJK15, CSK13, ZWC$^+$16]. Anonymity means that ratings of honest users are indistinguishable, whereas public linkability requires that anyone can decide whether or not two ratings for the same product were created by the same user. Also, ratings need to be traceable: the identity of any rater can be determined by a designated System Manager. In the course of this non-frameability guarantees that honest parties are not blamed having rated some product, when they did not. The combination of traceability and non-frameability enables penalizing dishonest behavior.

All previously mentioned works consider reputation systems in isolation, although reputation systems are always used in combination with other applications. In such situations stand-alone security definitions, as in [BJK15], do not guarantee security. With the Universal Composability Framework (UC) [Can01] there exists a methodology that guarantees security even in composed applications. Informally, in UC the execution of a real-life protocol is compared to the execution of an ideal protocol. If the real-life and ideal protocol executions are indistinguishable, then the real-life protocol is *UC-secure*. Based on this security definition Canetti [Can01] formulates a composition theorem which states that any UC-secure protocol is also secure when it is composed with other protocols.

Our Contribution

We present an ideal functionality for reputation systems $\mathcal{F}_{\mathsf{RS}}$ in the Universal Composability Framework [Can01]. Our ideal functionality prevents all previously mentioned attacks and provides anonymity, public linkability, traceability, and non-frameability. In contrast to [BJK15], users can rate each others products; there is no separation of customers and providers.

Besides defining an ideal functionality we present an efficient protocol for reputation systems that realizes $\mathcal{F}_{\mathsf{RS}}$. This protocol is influenced by different techniques known from Σ-protocols [Dam02] and (dynamic) group signatures [ACHdM05, BMW03, BSZ05, BBS04], similarly to the scheme in [BJK15]. But our protocol is more efficient and more flexible than the scheme in [BJK15] and it is secure even under concurrent composition (UC-secure).

2 The Ideal Functionality for Reputation Systems

In the first part of this section, we give some intuition to our ideal functionality of a reputation system $\mathcal{F}_{\mathsf{RS}}$. The second part concerns the formal definition of $\mathcal{F}_{\mathsf{RS}}$ in the Universal Composability Framework [Can01]. We discuss the functionality and its security properties in the third part of the section.

2.1 Intuition to our Reputation System

A meaningful reputation system must provide trustworthy, reliable, and honest ratings. Furthermore, it should be flexible in the sense that it can be combined with many different applications.

Therefore, we focus on the process of secure rating and provide a scheme that can be combined with any high-level application. For this reason, the aggregation of ratings and the evaluation of a specific reputation function is excluded from our model. Specifically, we handle the actual rating-message as a placeholder for the higher level application.

We consider reputation systems where users within the system can rate each others products. The term *product* refers to anything that can be used as a basis for ratings. Each user in our system has to *register* once at a *System Manager*, before a product can be rated. This prevents Sybil attacks, whitewashing attacks, bad mouthing attacks, and ballot stuffing attacks, and gives the System Manager the ability to punish misbehaving users. For this to work the system must prevent users to register with different identities. When users do not want to rate other products, a registration is not necessary - publishing products and verifying ratings is independent of the registration, which increases trust in the system. Analogously to registering, a product must be *purchased* prior to rating. This requirement assures that ratings are only given by raters using the product. Also, this is a protection mechanism against value imbalance attacks.

To further increase trust in the reputation system, raters must be able to rate purchased products anonymously. Without anonymity raters may tend to rate dishonestly when they fear negative consequences from the product owner. At the same time a product owner must be protected against unjustified negative ratings. This is achieved by giving the System Manager the ability to revoke the anonymity of a rater. Of course, the System Manager must not be able to accuse an honest user having misbehaved.

The negative side-effects of anonymity are that self-ratings, i.e. ratings for a product from the product owner, are hard to prevent and that a single rater who purchased a product could rate this product multiple times. Therefore we require a reputation system to explicitly forbid self-ratings and to provide *linkable ratings*: everybody - even outsiders of the system - must be able to detect multiple ratings from the same user for the same product.

As pointed out above, the security requirements a reputation system has to fulfill include - but are not limited to - *anonymity* for raters, *unforgeability* and *public linkability* of ratings, and the ability to determine the raters' identity. These properties have already been studied in the simpler context of group signatures [BMW03, BSZ05, BBS04, ACHdM05, FS07]. However, reputation systems have more security requirements than group signatures, as they do not consist of a single group of users. Instead, reputation systems can be seen as a collection of multiple group signature schemes - one for each product. Moreover, a single user may offer several products. Hence, in the definition of security properties the different group signature schemes must be considered in conjunction. Therefore, we adapt and extend these notions and give our formal definition of a secure reputation system in the Universal Composability Framework [Can01]. This framework guarantees security even for concurrently composed protocols. Stand-alone security definitions do not provide this strong guarantees, which are very important for our reputation system, as we intend it to be combined with other applications.

Additionally to the experiment-based security definitions for reputation systems [BJK15] and group signatures [BMW03, BSZ05], our ideal functionality $\mathcal{F}_{\mathsf{RS}}$ is influenced by the ideal functionalities for digital signatures $\mathcal{F}_{\mathsf{SIG}}$ [Can04], public-key encryption $\mathcal{F}_{\mathsf{PKE}}$ [Can01] and group signatures [ACHdM05].

2.1.1 The Universal Composability Framework

In contrast to stand-alone security definitions (both experiment-based and simulation-based), the Universal Composability Framework, introduced by Canetti [Can01], provides security under concurrent composition of different applications. To achieve this strong security notion, the execution of a real-life protocol is compared to the execution of an ideal protocol. Both protocol executions are controlled by an environment $\mathcal{Z}$ that tries to distinguish whether it interacts with the real-life protocol or the ideal protocol.

The ideal protocol is described by an ideal functionality $\mathcal{F}$ that handles every (cryptographic) task as a trusted party and interacts with an ideal adversary $\mathcal{S}$ (also called a simulator) and all parties involved in the protocol. Every party hands its inputs from the environment securely to $\mathcal{F}$. Then $\mathcal{F}$ computes the parties' output and sends it back to the party. Whenever a party receives a message from $\mathcal{F}$, the party outputs this message directly to the environment. The ideal adversary $\mathcal{S}$ may corrupt some parties and can block the delivery of messages from $\mathcal{F}$ to a party. The inputs a party hands to $\mathcal{F}$ cannot be seen by $\mathcal{S}$. In the real-life execution all parties compute their outputs by running the defined protocol. Analogously to $\mathcal{S}$, a real-life adversary $\mathcal{A}$ may corrupt parties within the real-life protocol execution.

We say that the real-life protocol UC-realizes the ideal protocol, if no environment can distinguish an interaction with the real-life protocol and $\mathcal{A}$ from an interaction with the ideal protocol and $\mathcal{S}$. Based on this security definition Canetti [Can01] formulates a composition theorem which states that any UC-secure protocol is also secure when it is executed concurrently with other protocols.

For our proof of security we will consider black-box simulators $\mathcal{S}$, denoted by $\mathcal{S}^{\mathcal{A}}$, that have block-box access to real-life adversaries $\mathcal{A}$. Also we consider a model with ideally authenticated channels, meaning that an adversary is able to read the messages sent, but is unable to modify them. We refer to this communication model as the *authenticated channels assumption.*

2.2 The Formal Definition of $\mathcal{F}_{\mathsf{RS}}$

Our ideal functionality interacts with the parties $P_{\mathrm{IDM}}, P_1, P_2, \ldots, P_n$ and an ideal adversary $\mathcal{S}$, which is also called a *simulator.* The party P_{IDM} acts as the System Manager, whereas the parties P_i correspond to the users within the reputation system. Furthermore, $\mathcal{F}_{\mathsf{RS}}$ manages the lists $\mathfrak{Params}, \mathfrak{Reg}, \mathfrak{Prods}, \mathfrak{Purch}, \mathfrak{Ratings}$, and $\mathfrak{Open}$ to store important information. Before giving the formal definition of $\mathcal{F}_{\mathsf{RS}}$, we explain how these lists are used. We also introduce the notation needed in the definition of $\mathcal{F}_{\mathsf{RS}}$.

$\mathfrak{Params}$: This list stores all pairs of the form (P_{IDM}, pp) containing *public parameters* the simulator $\mathcal{S}$ gives to $\mathcal{F}_{\mathsf{RS}}$ during KeyGen-requests. The first component of a pair is fixed to P_{IDM}, whereas the second component represents the actual parameters given by $\mathcal{S}$.

$\mathfrak{Reg}$: The list $\mathfrak{Reg}$ stores pairs of the form (pp, P_i) containing *registration information.* The first component stores the public parameters the registrated party used in the $\mathsf{Register}$-protocol, whereas the second component is the registrated party.

$\mathfrak{Prods}$: All *products* that are used within the reputation system are stored as 4-tuples (P_i, *prod*, *ppk*, b) in the list $\mathfrak{Prods}$. The first component of a tuple declares the product owner, the second is a product identifier (a bitstring chosen by the environment), the third specifies the corresponding *product-public key* and the fourth component is a validity bit. There can exist different products with the same product identifier, but for different product owners. The validity bit indicates whether the product-public key matches the given product owner and the product identifier.

$\mathfrak{Purch}$: When some party successfully purchased a product, this information is stored as 4-tuple $(P_i, P_j, prod, ppk)$ in the list $\mathfrak{Purch}$. For every tuple in the list the first component represents the purchaser, whereas the other components determine the product that was purchased (the product owner, the product identifier and the product-public key).

$\mathfrak{Ratings}$: The list $\mathfrak{Ratings}$ stores the most complex information as 10-tuples of the form (*pp*, P_i, P_j, *prod*, *ppk*, m, σ, b, *lid*, *oid*). The components of each tuple represent the following information:

1. *pp* - the public parameters a rating is generated for,
2. P_i - the identity of the rater ((pp, P_i) should match an entry in $\mathfrak{Reg}$),
3. P_j - the product owner of the product the rating is generated for,
4. *prod* - the product identifier of the product the rating is generated for,
5. *ppk* - the product-public key of the product the rating is generated for (the tuple (P_i, P_j, *prod*, *ppk*) should match an entry in $\mathfrak{Purch}$),
6. m - rating message (a placeholder for high-level applications),
7. σ - the rating,
8. b - the validity bit (indicating whether the rating is *valid*),
9. *lid* - the *linking-class identifier*, which is managed by the algorithm RebLDB, and
10. *oid* - the *opening-proof identifier*.

The linking-class identifier is needed to model the linkability property: two ratings with the same linking-class identifier have the same author. The opening-class identifier binds a list of opening-proofs to a specific rating. Whenever a new rating is added to the list $\mathfrak{Ratings}$, $\mathcal{F}_{\mathsf{RS}}$ uses the current value of a global counter *lidc* as the linking-class identifier and increments the counter. The subsequent execution of RebLDB ensures that the rating is put into the correct linking-class, according to the linkability-relation. A more detailed explanation of this behavior and the *oid*-mechanism is given in the discussion of the security properties of $\mathcal{F}_{\mathsf{RS}}$.

$\mathfrak{Open}$: This list stores all *opening-proofs* as 4-tuples of the form (oid, τ, b, P). The first component is an opening-proof identifier that binds a tuple to a specific rating with the same identifier. The second component is the actual opening-proof. The third component is a validity bit indicating whether the proof is *valid* and the fourth component is the claimed party that shall be the author of the associated rating. The value $oid = \perp$ within a rating expresses that the rating was not opened yet and hence no opening-proof exists. To uniquely bind opening-proofs to ratings a global counter *oidc* is used and incremented whenever a new opening-proof is bound to an unopened rating.

To manipulate the described lists, we introduce two operations:

- adding a tuple v to a list L is expressed by $L.\mathsf{Add}(v)$, and
- substituting a tuple v_{old} with a tuple v_{new} is expressed by $L.\mathsf{Sub}(v_{\text{old}}, v_{\text{new}})$.

Substituting a tuple v_{old} means that this tuple is removed from the list, while the tuple v_{new} is added to the list.

The classical notation to address components of tuples is using indices, i.e. $v = (v_1, v_2, \ldots, v_n)$, where v_i is the i'th component of tuple v. We deviate from this notation to prevent confusion with different variables and address the i'th component of a tuple v by $v[i]$.

Whenever $\mathcal{F}_{\mathsf{RS}}$ misses some information, the symbol $\perp$ is used to highlight this fact. Also the simulator $\mathcal{S}$ can output this symbol at some points to indicate that it is not able to respond to a request. Depending on the situation, this is not necessarily a failure.

With these prerequisites we now give the formal definition of $\mathcal{F}_{\mathsf{RS}}$.

$\mathcal{F}_{\mathsf{RS}}$

$\mathcal{F}_{\mathsf{RS}}$ interacts with parties $P_{\text{IDM}}, P_1, \ldots, P_n$, and the ideal adversary (simulator) $\mathcal{S}$. Further it manages the lists $\mathfrak{Params}, \mathfrak{Reg}, \mathfrak{Prods}, \mathfrak{Purch}, \mathfrak{Ratings}$, and $\mathfrak{Open}$ which are initially empty, and the counters *lidc*, *oidc*, which are initialized with 0. All outputs from $\mathcal{F}_{\mathsf{RS}}$ to some party P are public delayed outputs.

Registry Key Generation: On input $(\mathsf{KeyGen}, \mathit{sid})$ from P_{IDM}

```
Check that $\mathit{sid} = (P_{\text{IDM}}, \mathit{sid}')$ for some $\mathit{sid}'$. If not, ignore the request.
Send $(\mathsf{KeyGen}, \mathit{sid})$ to $\mathcal{S}$ and receive $(\mathsf{KeyGen}, \mathit{sid}, \mathit{pp})$ from $\mathcal{S}$.
Set $\mathfrak{Params}.\mathsf{Add}(P_{\text{IDM}}, \mathit{pp})$ and send $(\mathsf{KeyGen}, \mathit{sid}, \mathit{pp})$ to $P_{\text{IDM}}$.
```

User Registration: On input $(\mathsf{Register}, \mathit{sid}, \mathit{pp}')$ from P_i

```
Check that $\mathit{sid} = (P_{\text{IDM}}, \mathit{sid}')$ for some $\mathit{sid}'$. If not, ignore the request.
Send $(\mathsf{Register}, \mathit{sid}, \mathit{pp}', P_i)$ to $\mathcal{S}$ and receive $(\mathsf{Register}, \mathit{sid}, \mathit{pp}', P_i, b)$ from $\mathcal{S}$.
If $P_{\text{IDM}}$ and $P_i$ are honest $\wedge$ $(P_{\text{IDM}}, \mathit{pp}') \in \mathfrak{Params}$ $\wedge$ $(P_i, \mathit{pp}') \notin \mathfrak{Reg}$ then $f := 1$.
Else If $P_{\text{IDM}}$ is honest $\wedge$ $(P_{\text{IDM}}, \mathit{pp}') \notin \mathfrak{Params}$ then $f := 0$.
Else $f := b$.
If $f = 1$ then $\mathfrak{Reg}.\mathsf{Add}(\mathit{pp}', P_i)$.
Send $(\mathsf{Register}, \mathit{sid}, \mathit{pp}', P_i, f)$ to $P_i$ and $P_{\text{IDM}}$.
```

Product Addition: On input $(\mathsf{NewProduct}, \mathit{sid}, \mathit{prod})$ from P_i

```
Check that $\mathit{sid} = (P_{\text{IDM}}, \mathit{sid}')$ for some $\mathit{sid}'$. If not, ignore the request.
Send $(\mathsf{NewProduct}, \mathit{sid}, P_i, \mathit{prod})$ to $\mathcal{S}$ and receive $(\mathsf{NewProduct}, \mathit{sid}, P_i, \mathit{prod}, \mathit{ppk})$ from $\mathcal{S}$.
```

```
If $(P', \mathit{prod}', \mathit{ppk}, 1) \in \mathfrak{Prods}$, where $(P', \mathit{prod}') \neq (P_i, \mathit{prod})$ then output $\texttt{error}$ and halt.
Else $\mathfrak{Prods}.\mathsf{Add}(P_i, \mathit{prod}, \mathit{ppk}, 1)$ and send $(\mathsf{NewProduct}, \mathit{sid}, \mathit{prod}, \mathit{ppk})$ to $P_i$.
```

Purchase: On input $(\mathsf{Purchase}, \mathit{sid}, P_j, \mathit{prod}, \mathit{ppk})$ from P_i

```
Check that $\mathit{sid} = (P_{\mathrm{IDM}}, \mathit{sid}')$ for some $\mathit{sid}'$. If not, ignore the request.
If $P_i = P_j \vee \mathsf{VfyProd}(\mathit{sid}, P_j, \mathit{prod}, \mathit{ppk}) = 0$ then ignore the request.
Send $(\mathsf{Purchase}, \mathit{sid}, P_i, P_j, \mathit{prod}, \mathit{ppk})$ to $\mathcal{S}$ and receive $(\mathsf{Purchase}, \mathit{sid}, P_i, P_j, \mathit{prod}, \mathit{ppk}, b)$ from $\mathcal{S}$.
If $P_i$ and $P_j$ are honest then $f := 1$.
Else $f := b$.
If $f = 1$ then $\mathfrak{Purch}.\mathsf{Add}(P_i, P_j, \mathit{prod}, \mathit{ppk})$.
Send $(\mathsf{Purchase}, \mathit{sid}, P_i, P_j, \mathit{prod}, \mathit{ppk}, f)$ to $P_i$ and $P_j$.
```

$\mathsf{VfyProd}$: On internal input $(\mathit{sid}, P_j, \mathit{prod}, \mathit{ppk})$

```
Send $(\mathsf{VfyProd}, \mathit{sid}, P_j, \mathit{prod}, \mathit{ppk})$ to $\mathcal{S}$ and receive $(\mathsf{VfyProd}, \mathit{sid}, P_j, \mathit{prod}, \mathit{ppk}, b)$ from $\mathcal{S}$.
If $(P_j, \mathit{prod}, \mathit{ppk}, f') \in \mathfrak{Prods}$ then $f := f'$.
Else If $b = 1 \wedge P_j$ is honest then output $\texttt{error}$ and halt.
Else If $(P', \mathit{prod}', \mathit{ppk}, 1) \in \mathfrak{Prods}$, where $(P', \mathit{prod}') \neq (P_i, \mathit{prod})$ then
        $\mathfrak{Prods}.\mathsf{Add}(P_i, \mathit{prod}, \mathit{ppk}, 0)$ and $f := 0$.
Else Set $\mathfrak{Prods}.\mathsf{Add}(P_i, \mathit{prod}, \mathit{ppk}, b)$ and $f := b$.
Return $f$.
```

Rate a Product: On input $(\mathsf{Rate}, \mathit{sid}, \mathit{pp}, P_j, \mathit{prod}, \mathit{ppk}, m)$ from P_i

```
Check that $\mathit{sid} = (P_{\mathrm{IDM}}, \mathit{sid}')$ for some $\mathit{sid}'$. If not, ignore the request.
If $(\mathit{pp}, P_i) \notin \mathfrak{Reg} \vee (P_i, P_j, \mathit{prod}, \mathit{ppk}) \notin \mathfrak{Purch} \vee (\mathit{pp}, P_i, P_j, \mathit{prod}, \mathit{ppk}, m', \sigma', 1, \mathit{lid}, \mathit{oid}) \in \mathfrak{Ratings}$ for some $m', \sigma', \mathit{lid}$ then ignore the request.
If $P_{\mathrm{IDM}}$ is honest then
    Send $(\mathsf{Rate}, \mathit{sid}, \mathit{pp}, P_j, \mathit{prod}, \mathit{ppk}, m)$ to $\mathcal{S}$ and receive $(\mathsf{Rate}, \mathit{sid}, \mathit{pp}, P_j, \mathit{prod}, \mathit{ppk}, m, \sigma)$ from $\mathcal{S}$.
Else Send $(\mathsf{Rate}, \mathit{sid}, \mathit{pp}, P_i, P_j, \mathit{prod}, \mathit{ppk}, m)$ to $\mathcal{S}$ and receive $(\mathsf{Rate}, \mathit{sid}, \mathit{pp}, P_i, P_j, \mathit{prod}, \mathit{ppk}, m, \sigma)$ from $\mathcal{S}$.
If $(\mathit{pp}, P', P_j, \mathit{prod}, \mathit{ppk}, m, \sigma, 0, \mathit{lid}, \mathit{oid}) \in \mathfrak{Ratings}$ for some $P', \mathit{lid}, \mathit{oid}$ then
    Output $\texttt{error}$ and halt.
Set $r := (\mathit{pp}, P_i, P_j, \mathit{prod}, \mathit{ppk}, m, \sigma, 1, \mathit{lidc}, \perp)$ and $\mathit{lidc} := \mathit{lidc} + 1$.
Set $\mathfrak{Ratings}.\mathsf{Add}(r)$ and run $\mathsf{RebLDB}(\mathit{sid}, r, \perp)$.
Send $(\mathsf{Rate}, \mathit{sid}, \mathit{pp}, P_j, \mathit{prod}, \mathit{ppk}, m, \sigma)$ to $P_i$.
```

Verifying a Rating: On input $(\mathsf{Verify}, \mathit{sid}, \mathit{pp}, P_j, \mathit{prod}, \mathit{ppk}, m, \sigma)$ from P_i

```
Check that $\mathit{sid} = (P_{\mathrm{IDM}}, \mathit{sid}')$ for some $\mathit{sid}'$. If not, ignore the request.
Set $(X, f, \mathit{oid}) := \mathsf{VfyRtg}(\mathit{sid}, \mathit{pp}, P_j, \mathit{prod}, \mathit{ppk}, m, \sigma)$.
Send $(\mathsf{Verify}, \mathit{sid}, \mathit{pp}, P_j, \mathit{prod}, \mathit{ppk}, m, \sigma, f)$ to $P_i$.
```

```
VfyRtg: On internal input $(sid, pp, P_j, prod, ppk, m, \sigma)$
  If $\mathsf{VfyProd}(sid, P_j, prod, ppk) = 0$ then ignore the request.
  Send $(\mathsf{Verify}, sid, pp, P_j, prod, ppk, m, \sigma)$ to $\mathcal{S}$ and receive $(\mathsf{Verify}, sid, pp, P_j, prod, ppk, m, \sigma, b, P)$ from $\mathcal{S}$.
  If $(pp, X', P_j, prod, ppk, m, \sigma, f', lid', oid') \in \mathfrak{Ratings}$ for some $X', f', lid'$ and $oid'$ then
      $X := X', f := f', oid := oid'$.
  Else If $b = 0 \vee P = P_j$ then
      Set $\mathfrak{Ratings}.\mathsf{Add}(pp, \bot, P_j, prod, ppk, m, \sigma, 0, \bot, \bot)$, $X := \bot$, $f := 0$, and $oid := \bot$.
  Else If $P_j$ is honest, $P \neq \bot$ and $(P, P_j, prod, ppk) \notin \mathfrak{Purch}$ then Output error and halt.
  Else If $P \neq \bot$ and $P$ is honest then output error and halt.
  Else If $P = \bot$ and $P_{\mathrm{IDM}}$ is honest then output error and halt.
  Else Set $r := (pp, P, P_j, prod, ppk, m, \sigma, 1, lidc, \bot)$, $X := P$, $f := 1$, and $oid := \bot$.
      Set $lidc := lidc + 1$, $\mathfrak{Ratings}.\mathsf{Add}(r)$, and run $\mathsf{RebLDB}(sid, r, \bot)$.
  Return $(X, f, oid)$.
```

```
Linking Ratings: On input $(\mathsf{Link}, sid, pp, P_j, prod, ppk, m_0, \sigma_0, m_1, \sigma_1)$ from $P_i$
  Check that $sid = (P_{\mathrm{IDM}}, sid')$ for some $sid'$. If not, ignore the request.
  Set $b := \mathsf{LinkRtgs}(sid, pp, P_j, prod, ppk, m_0, \sigma_0, m_1, \sigma_1)$.
  Send $(\mathsf{Link}, sid, pp, P_j, prod, ppk, m_0, \sigma_0, m_1, \sigma_1, b)$ to $P_i$.
```

```
LinkRtgs: On internal input $(sid, pp, P_j, prod, ppk, m_0, \sigma_0, m_1, \sigma_1)$
  Set $(X_k, f_k, oid_k) := \mathsf{VfyRtg}(sid, pp, P_j, prod, ppk, m_k, \sigma_k)$ for $k \in \{0, 1\}$.
  Send $(\mathsf{Link}, sid, pp, P_j, prod, ppk, m_0, \sigma_0, m_1, \sigma_1)$ to $\mathcal{S}$, receive $(\mathsf{Link}, sid, pp, P_j, prod, ppk, m_0, \sigma_0, m_1, \sigma_1, b)$ from $\mathcal{S}$, and set $f := 0$.
  If $f_0 = f_1 = 1$ then
      Let $r_k \in \mathfrak{Ratings}$ be the unique tuples $r_k := (pp, X_k, P_j, prod, ppk, m_k, \sigma_k, 1, lid_k, oid_k)$, for $k \in \{0, 1\}$.
      If $lid_0 = lid_1$ then $f := 1$.
      Else If $X_0 = X_1 \wedge X_0 = \bot \wedge X_1 = \bot$ then $f := b$.
      Else If $X_0 \neq X_1 \wedge X_0 \neq \bot \wedge X_1 \neq \bot$ then $f := 0$.
      Else If $(X_k = \bot \wedge X_{1-k} \neq \bot \wedge X_{1-k}$ is honest) for $k = 0 \vee k = 1$ then $f := 0$.
      Else If $(X_k = \bot \wedge X_{1-k} \neq \bot \wedge X_{1-k}$ is corrupted) for $k = 0 \vee k = 1$ then $f := b$.
      If $f = 1$ then run $\mathsf{RebLDB}(sid, r_0, r_1)$.
  Return $f$.
```

```
RebLDB: On internal input $(sid, r, s)$
  Parse $r$ as $(pp, X_0, P_j, prod, ppk, m_0, \sigma_0, 1, lid_0, oid_0)$.
  If $s = \bot \wedge X_1 \neq \bot$ then
      Set $\mathcal{L} := \{\ell | \ell \in \mathfrak{Ratings} \wedge \ell[1] = pp \wedge \ell[2] = X_0 \wedge \ell[3] = P_j \wedge \ell[4] = prod \wedge \ell[5] = ppk \wedge \ell[8] = 1\}$ and $lid := \min\{\ell[9] | \ell \in \mathcal{L}\}$.
      For each $\ell \in \mathcal{L}$ do Set $\ell' := \ell$, $\ell'[9] := lid$, and $\mathfrak{Ratings}.\mathsf{Sub}(\ell, \ell')$.
```

```
If $s \neq \bot$ then
    Parse $s$ as $(pp, X_1, P_j, prod, ppk, m_1, \sigma_1, 1, lid_1, oid_1)$
    If $X_0 = \bot \wedge X_1 \neq \bot$ then Set $X \coloneqq X_1$.
    Else Set $X \coloneqq X_0$.
    Set $\mathcal{L} \coloneqq \{\ell | \ell \in \mathfrak{Ratings} \wedge \ell[1] = pp \wedge \ell[3] = P_j \wedge \ell[4] = prod \wedge \ell[5] = ppk \wedge \ell[8] = 1 \ \wedge (\ell[9] = lid_0 \vee \ell[9] = lid_1)\}$ and $lid \coloneqq \min\{lid_0, lid_1\}$.
    For each $\ell \in \mathcal{L}$ do Set $\ell' \coloneqq \ell$, $\ell'[2] \coloneqq X$, $\ell'[9] \coloneqq lid$, and $\mathfrak{Ratings}.\mathsf{Sub}(\ell, \ell')$.
Set $\mathcal{P} \coloneqq \{p | p \in \mathfrak{Purch} \wedge p[2] = P_j \wedge p[3] = prod \wedge p[4] = ppk\}$.
Set $\mathcal{L} \coloneqq \{\ell | \ell \in \mathfrak{Ratings} \wedge \ell[1] = pp \wedge \ell[3] = P_j \wedge \ell[4] = prod \wedge \ell[5] = ppk \wedge \ell[8] = 1\}$.
If $P_{\mathrm{IDM}}$ is corrupted, $P_j$ is honest and $|\mathcal{P}| < |\{\ell[9] | \ell \in \mathcal{L}\}|$ then
    For each $(\ell, \ell') \in \mathcal{L}^2$ do
            Run $\mathsf{LinkRtgs}(sid, \ell[1], \ell[3], \ell[4], \ell[5], \ell[6], \ell[7], \ell'[6], \ell'[7])$ ignoring the output of $\mathsf{LinkRtgs}$.
    Set $\mathcal{P} \coloneqq \{p | p \in \mathfrak{Purch} \wedge p[2] = P_j \wedge p[3] = prod \wedge p[4] = ppk\}$.
    Set $\mathcal{L} \coloneqq \{\ell | \ell \in \mathfrak{Ratings} \wedge \ell[1] = pp \wedge \ell[3] = P_j \wedge \ell[4] = prod \wedge \ell[5] = ppk \wedge \ell[8] = 1\}$.
If $P_j$ is honest and $|\mathcal{P}| < |\{\ell[9] | \ell \in \mathcal{L}\}|$ then Output error and halt.
```

Determine Raters Identity: On input $(\mathsf{Open}, sid, pp, P_j, prod, ppk, m, \sigma)$ from P_{IDM}

```
Check that $sid = (P_{\mathrm{IDM}}, sid')$ for some $sid'$. If not, ignore the request.
If $(P_{\mathrm{IDM}}, pp) \notin \mathfrak{Params}$ then ignore the request.
Set $(X, f, oid) \coloneqq \mathsf{VfyRtg}(sid, pp, P_j, prod, ppk, m, \sigma)$.
If $f = 1$ then
    Let $r \in \mathfrak{Ratings}$ be the unique tuple $r \coloneqq (pp, X, P_j, prod, ppk, m, \sigma, 1, lid', oid)$ for some $lid'$.
    If $oid = \bot$ then Set $r' \coloneqq r$, $r'[10] \coloneqq oidc$, $\mathfrak{Ratings}.\mathsf{Sub}(r, r')$ and $oidc \coloneqq oidc + 1$.
    Send $(\mathsf{Open}, sid, pp, P_j, prod, ppk, m, \sigma, X)$ to $P_{\mathrm{IDM}}$.
Else Send $(\mathsf{Open}, sid, pp, P_j, prod, ppk, m, \sigma, \bot)$ to $P_{\mathrm{IDM}}$.
```

Generate Opening Proofs: On input $(\mathsf{OProof}, sid, pp, P_j, prod, ppk, m, \sigma, P)$ from P_{IDM}

```
Check that $sid = (P_{\mathrm{IDM}}, sid')$ for some $sid'$. If not, ignore the request.
If $(P_{\mathrm{IDM}}, pp) \notin \mathfrak{Params}$ then ignore the request.
Set $(X, f, oid) \coloneqq \mathsf{VfyRtg}(sid, pp, P_j, prod, ppk, m, \sigma)$.
Send $(\mathsf{OProof}, sid, pp, P_j, prod, ppk, m, \sigma, P)$ to $\mathcal{S}$ and receive $(\mathsf{OProof}, sid, pp, P_j, prod, ppk, m, \sigma, P, \tau)$ from $\mathcal{S}$.
If $f \neq 1 \vee X \neq P \vee oid = \bot$ then Send $(\mathsf{OProof}, sid, pp, P_j, prod, ppk, m, \sigma, P, \bot)$ to $P_{\mathrm{IDM}}$.
Else
    If $\tau = \bot \vee (oid, \tau, 0, P) \in \mathfrak{Open}$ then output error and halt.
    $\mathfrak{Open}.\mathsf{Add}(oid, \tau, 1, P)$ and send $(\mathsf{OProof}, sid, pp, P_j, prod, ppk, m, \sigma, P, \tau)$ to $P_{\mathrm{IDM}}$.
```

Verifying Opening-Proofs: On input $(\mathsf{Judge}, sid, pp, P_j, prod, ppk, m, \sigma, P, \tau)$ from P_i

```
Check that $sid = (P_{\mathrm{IDM}}, sid')$ for some $sid'$. If not, ignore the request.
```

```
Set $(X, f, oid) := \mathsf{VfyRtg}(sid, pp, P_j, prod, ppk, m, \sigma)$.
Send $(\mathsf{Judge}, sid, pp, P_j, prod, ppk, m, \sigma, P, \tau)$ to $\mathcal{S}$, receive $(\mathsf{Judge}, sid, pp, P_j, prod, ppk, m, \sigma, P, \tau, b)$ from $\mathcal{S}$, and set $v := b$.
If $f = 0 \lor P = \perp \lor \tau = \perp$ then Send $(\mathsf{Judge}, sid, pp, P_j, prod, ppk, m, \sigma, P, \tau, 0)$ to $P_i$.
Else If $X \neq \perp$ then
        Let $r \in \mathfrak{Ratings}$ be the unique tuple $r := (pp, X, P_j, prod, ppk, m, \sigma, 1, lid', oid)$ for some $lid'$, and set $r' := r$.
        If $X = P \land (oid, \tau, 1, P) \in \mathfrak{Open}$ then $v := 1$.
        Else If $X \neq P \lor (oid, \tau, 0, P) \in \mathfrak{Open}$ then $v := 0$.
        Else If $P_{\mathrm{IDM}}$ and $P$ are honest and $b = 1$ then output error and halt.
Else
    Let $r \in \mathfrak{Ratings}$ be the unique tuple $r := (pp, \perp, P_j, prod, ppk, m, \sigma, 1, lid', oid)$ for some $lid'$, and set $r' := r$.
    If $(oid, \tau, 0, P) \in \mathfrak{Open}$ then $v := 0$.
    Else If $b = 1 \land P$ is honest then output error and halt.
    Else If $b = 1$ then set $v := 1$, $r'[2] := P$, $\mathfrak{Ratings}.\mathsf{Sub}(r, r')$, $r := r'$
            Run $\mathsf{RebLDB}(sid, r', \perp)$.
If $oid = \perp$ then $r'[10] := oidc$, $\mathfrak{Ratings}.\mathsf{Sub}(r, r')$, $\mathfrak{Open}.\mathsf{Add}(oidc, \tau, v, P)$, $oidc := oidc + 1$.
Else $\mathfrak{Open}.\mathsf{Add}(oid, \tau, v, P)$.
Send $(\mathsf{Judge}, sid, pp, P_j, prod, ppk, m, \sigma, P, \tau, v)$ to $P_i$.
```

Functionality 1: Reputation System

2.2.1 Security Properties of $\mathcal{F}_{\mathsf{RS}}$

As many other ideal functionalities in the UC framework, we define $\mathcal{F}_{\mathsf{RS}}$ to work as a "registy service" to store parameters, ratings, and opening-proofs. Using the right parameters, every party is able to check whether ratings and opening-proofs are stored by $\mathcal{F}_{\mathsf{RS}}$. In all activations, $\mathcal{F}_{\mathsf{RS}}$ lets the simulator $\mathcal{S}$ choose the values needed to respond to the activation. The requirements on these values are defined as restrictions for each activation. In the following, we discuss these restrictions and the implied security properties.

Registry Key Generation: Similar to the Signature Functionality $\mathcal{F}_{\mathsf{SIG}}$ [Can04] and the Public-Key Encryption Functionality $\mathcal{F}_{\mathsf{PKE}}$ [Can01], we do not make any security relevant requirements on the *public parameters pp*.

User Registration: Being registered is a prerequisite to rate a product and covers the first step to prevent Sybil attacks, whitewashing attacks, bad mouthing attacks, and ballot stuffing attacks. The user registration models an interactive protocol between P_{IDM} and some party P_i. In general, $\mathcal{F}_{\mathsf{RS}}$ lets the simulator $\mathcal{S}$ decide whether party P_i successfully registered, with the following two restrictions: non-registered honest parties communicating with an honest P_{IDM} using the right public parameters will always be registered after the protocol execution ($b = 1$) and an honest P_{IDM} will reject a party from registering, when wrong parameters are used ($b = 0$).

Product Addition and $\mathsf{VfyProd}$: The $\mathsf{NewProduct}$-activation is used by party P_i to publish a new *product-public key ppk* for a given product $prod \in \{0,1\}^*$. The value *ppk* is bound to the bitstring *prod* and to the party requesting it, such that every party can validate the ownership of a product. Formally this means, that a product-public key is only *valid* for one specific pair $(P, prod)$. This is a very important requirement, because it models *unforgeability* of product-public keys. Without this property any corrupted party P_j could "copy" some *ppk* (that was generated by an honest party P_i) and declare foreign ratings as own ratings: all valid ratings for $(P_i, prod, ppk)$ would also be valid for $(P_j, prod', ppk')$. Since we want to have a reliable, trustworthy and fair system such attacks must be prevented. We emphasize that $\mathsf{VfyProd}$ is modeled as an internal subroutine within $\mathcal{F}_{\mathsf{RS}}$ and is implicitly used in other activations.

Purchase: Another prerequisite to rate a product is to purchase it. This is necessary to prevent value imbalance attacks. The purchasing protocol is an interactive protocol between two parties: the seller P_j and the purchaser P_i. Naturally, before purchasing a product its corresponding product-public key is verified. Only if this is valid, the protocol will be executed. For two honest parties the purchasing process will successfully finish, whereas the simulator $\mathcal{S}$ determines the outcome of the protocol execution in any other case.

Rating a Product: When party P_i wants to rate the product *prod* with public key *ppk* owned by party P_j, P_i must be registered, must have purchased the specified product, and must not have rated the product before. Being registered is necessary to *open* ratings, whereas having purchased the product enables rating verifiers to detect self-ratings, bad mouthing attacks and ballot stuffing attacks. In the case that P_{IDM} is honest, $\mathcal{F}_{\mathsf{RS}}$ guarantees *anonymity of raters*: the simulator $\mathcal{S}$ is asked to output a *rating* σ, that is valid for the specified product, without knowing the rating party. Hence, the output rating cannot depend on the raters' identity. In the case that P_{IDM} is corrupted, the simulator $\mathcal{S}$ obtains the identity of the rater, because in this case anonymity cannot be achieved.

Rating Verification and Determining the Raters' Identity: Given the right parameters, every rating can be verified. Note that ratings are only verified, if the specified product is valid. A *valid rating* guarantees the following properties, even for maliciously generated ratings:

- Non-Self-Rating: the rater is not the owner of the product.
- Linkability: the rater purchased the product (will be discussed later in detail).
- Traceability: the rater is registered and can be identified.

Every single property is crucial for trustworthy reputation. If self-ratings would not be prevented, ballot stuffing attacks were possible. The same holds for linkability, but this will be discussed later in detail. Being able to open ratings is also very important in practical applications, because otherwise misbehaving parties can not be identified and punished. Hence, it must be guaranteed that honest parties are not blamed having rated some product, when they did not. This property is called *non-frameability* and is discussed later in detail.

$\mathcal{F}_{\mathsf{RS}}$ not only asks the simulator $\mathcal{S}$ to validate a rating, but also to determine the raters' identity. This models the ability of P_{IDM} to open *every* rating, not only those for which an Open-request occurs. Furthermore, it simplifies the definition of $\mathcal{F}_{\mathsf{RS}}$ without weakening the security properties, because VfyRtg encapsulates all important characteristics of a valid rating in a single and reusable procedure.

Linking Ratings and RebLDB: For every party using a reputation system it is important to know whether two valid ratings for the same product are generated by the same party. If this is true, the rater behaved dishonestly. We call this property *linkability*, which prevents bad mouthing attacks and ballot stuffing attacks. Linkability represents an equivalence relation: $\mathsf{Link}(x, x) = 1$, $\mathsf{Link}(x, y) = \mathsf{Link}(y, x)$ and $\mathsf{Link}(x, y) = 1 \wedge \mathsf{Link}(y, z) = 1 \Rightarrow \mathsf{Link}(x, z) = 1$. The value *lid* stored by $\mathcal{F}_{\mathsf{RS}}$ for every rating represents the equivalence class the rating belongs to. Initially, *lid* is set to the current value of a global counter *lidc*. The linking-class identifiers are updated by the RebLDB algorithm whenever a new rating is added to the list $\mathfrak{Ratings}$ (via Rate and Verify) or new linking information is obtained (via Link and Judge). This algorithm is only for internal use and not callable by any party. The RebLDB-algorithm merges two equivalence classes in the following cases:

- Step 2 covers calls to the algorithm from Rate, Verify, and Judge ($s = \perp$), where P_{IDM} is not corrupted and/or X_1 is an uncorrupted rater ($X_1 \neq \perp$). In these cases RebLDB selects all valid ratings for the specified product from the same rater X_1 (the set $\mathcal{L}$) and sets the value *lid* ($\ell[9]$ for $\ell \in \mathcal{L}$) for all ratings in $\mathcal{L}$ to the minimal value within the selected ratings.
- Step 5 handles requests from Link where either the identity of the rater is not known but the simulator $\mathcal{S}$ tells $\mathcal{F}_{\mathsf{RS}}$ that these ratings are linkable (Step 6 of Link), or the identity of some corrupted party can be updated for some rating, because it is linkable to another rating $\mathcal{F}_{\mathsf{RS}}$ already knows the identity of (Step 9 in Link). According to the transitivity of the linkability relation, RebLDB merges the two equivalence classes into one class by selecting all ratings within the two classes (Step 9) and setting *lid* to be the smaller of both values. Additionally, if a party identity is given in X_1 or X_2 this value will be set for all ratings within the equivalence class (Step 10).
- In Steps RebLDB–18 RebLDB verifies that there do not exist more equivalence classes for an honestly generated product than the party owning the product sold. This ensures that it is only possible to rate a product once (without being linkable) after purchasing.

  When P_{IDM} is corrupted, it is possible that no linking information is available to $\mathcal{F}_{\mathsf{RS}}$. In this case $\mathcal{F}_{\mathsf{RS}}$ asks the simulator $\mathcal{S}$ to link all ratings for the product in question. Without this step a simple attack is possible:

  - $\mathcal{Z}$ lets the real-world adversary $\mathcal{A}$ corrupt P_{IDM} and some party P_i, lets P_i purchase some product from an honest party P_j, generates multiple valid ratings for this product and verifies them.
  - In this scenario $\mathcal{F}_{\mathsf{RS}}$ adds the ratings to $\mathfrak{Ratings}$ during the Verify-protocol, which in turn calls RebLDB. Since no linking information is available to $\mathcal{F}_{\mathsf{RS}}$, without

Step 13 $\mathcal{F}_{\mathsf{RS}}$ outputs `error`, even when all ratings are linkable. Hence, no protocol can realize $\mathcal{F}_{\mathsf{RS}}$.

If after Step 13 there are still more equivalence classes than purchases, this violates the security requirements of $\mathcal{F}_{\mathsf{RS}}$.

Summarizing, the handling of equivalence classes is modeled by the RebLDB-algorithm which uses linking information obtained from the algorithms Rate, Verify, Link, and Judge.

Generating and Verifying Opening-Proofs: Opening-proofs are values that enable every party to verify that a blamed party is really the author of a given rating. This covers the property of *non-frameability*: no honest party can be accused being the author of a given rating, when it is not. $\mathcal{F}_{\mathsf{RS}}$ asks the simulator $\mathcal{S}$ to output *valid* opening-proofs and ignores the output of $\mathcal{S}$, if the given rating is invalid, a wrong identity is given or the rating has not been opened yet. Since there can be more than one valid opening-proof, the value *oid* is used to connect a rating with its list of opening-proofs. This mechanism ensures that an opening-proof cannot be used to determine a raters identity for other ratings.

3 Realizing $\mathcal{F}_{\mathsf{RS}}$

Before introducing the protocol that realizes $\mathcal{F}_{\mathsf{RS}}$, we give the required preliminaries and building blocks in this section.

3.1 Preliminaries

Our realization relies on bilinear groups, the Symmetric External Diffie-Hellman-Assumption, and the Pointcheval-Sanders-Assumption. For completeness, we give the respective definitions in this section.

Definition 3.1 (Bilinear Groups)**.** A bilinear group $\mathbb{GD}$ is a set of three cyclic groups $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$, each group of prime order p, along with a bilinear map $\mathrm{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the following properties:

1. Bilinearity: for all $u \in \mathbb{G}_1$, $v \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p : \mathrm{e}(u^a, v^b) = \mathrm{e}(u, v)^{ab}$.
2. Non-degeneracy: for $u \neq 1_{\mathbb{G}_1}$ and $v \neq 1_{\mathbb{G}_2} : \mathrm{e}(u, v) \neq 1_{\mathbb{G}_T}$.
3. The map e is efficiently computable.

We will use pairings of Type-3 for our construction, because they allow efficient implementations and the Pointcheval-Sanders-Assumption does not hold in Type-1 and Type-2 pairing groups. Furthermore, for Type-3 pairing groups it is believed that the Decisional-Diffie-Hellman-Problem is hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$. This assumption is often referred to as the Symmetric External Diffie-Hellman-Assumption (SXDH) [GSW10].

Definition 3.2 (Bilinear Group Generator)**.** A bilinear group generator, denoted by BiGrGen, is a probabilistic polynomial time algorithm that, on input 1^λ, outputs a description of a bilinear group $\mathbb{GD}$. We denote the output of BiGrGen by $\mathbb{GD} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathrm{e}, g_1, g_2)$.

Definition 3.3 (Pointcheval-Sanders-Problem – PS1)**.** Let $\mathbb{GD} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathrm{e}, g_1, g_2)$ be a bilinear group setting of Type-3, with generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. Further, let $g \leftarrow \mathbb{G}_1$, $\tilde{g} \leftarrow \mathbb{G}_2$, $X := g^x$, $Y := g^y \in \mathbb{G}_1$ and $\tilde{X} := \tilde{g}^x$, $\tilde{Y} := \tilde{g}^y \in \mathbb{G}_2$, for $x, y \leftarrow \mathbb{Z}_p$. We define the oracle $\mathcal{O}(m)$ as follows: on input $m \in \mathbb{Z}_p$, choose $h \leftarrow \mathbb{G}_1$ and output $(h, h^{x+m\cdot y})$. Given $(g, Y, \tilde{g}, \tilde{X}, \tilde{Y})$ and unlimited access to oracle $\mathcal{O}$, the Pointcheval-Sanders-Problem is to output a tuple $(m^*, s, s^{x+m^*\cdot y})$, where $s \neq 1_{\mathbb{G}_1}$ and m^* was not asked to $\mathcal{O}$.

We say the Pointcheval-Sanders-Assumption holds for bilinear group generator $\mathsf{BiGrGen}$ if for all probabilistic polynomial time adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr\left[\mathcal{A}^{\mathcal{O}(\cdot)}\left(\mathbb{GD}, g, Y, \tilde{g}, \tilde{X}, \tilde{Y}\right) = \left(m^*, s, s^{x+m^*\cdot y}\right)\right] \leq \mathrm{negl}(\lambda),$$

where the probability is taken over the random bits used by $\mathsf{BiGrGen}, \mathcal{A}$, and the random choices of $x, y \leftarrow \mathbb{Z}_p$.

3.2 Building Blocks and Intuition for our Realization

In this section we briefly introduce the building blocks of our realization and explain how they are combined to realize $\mathcal{F}_{\mathsf{RS}}$.

We use *Pointcheval-Sanders Signatures* ($\mathsf{PS} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$) [PS16] as certificates for registration and for purchased products. We call the certificate for registration a *registration token*, the certificate for purchased products a *rating token*. To obtain such tokens every user has to prove knowledge of a self-chosen *user-secret-key usk*. We use the concurrent zero-knowledge variant of Σ-protocols, which uses *Trapdoor Pedersen Commitments* ($\mathsf{PD} = (\mathsf{KeyGen}, \mathsf{Commit}, \mathsf{Reveal}, \mathsf{Equiv})$) for this purpose.

Definition 3.4 (Pointcheval-Sanders Signatures (PS))**.** Let $\mathbb{GD} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathrm{e}, g_1, g_2)$ be a bilinear group setting of Type-3, with generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. The Pointcheval-Sanders Signature Scheme for messages $m \in \mathbb{Z}_p$ is defined as follows:

$\mathsf{KeyGen}(\mathbb{GD})$

1: Choose $\xi_1, \xi_2 \leftarrow \mathbb{Z}_p$, and $\tilde{g} \leftarrow \mathbb{G}_2$.
2: Set $(\tilde{X}, \tilde{Y}) := (\tilde{g}^{\xi_1}, \tilde{g}^{\xi_2})$, $sk := (\xi_1, \xi_2)$ and $pk := (\tilde{g}, \tilde{X}, \tilde{Y})$ and output (pk, sk).

$\mathsf{Sign}(sk, m)$

Choose $s \leftarrow \mathbb{G}_1$, set $\sigma := (\sigma_1, \sigma_2) := (s, s^{\xi_1+\xi_2\cdot m})$ and output σ as signature on m.

$\mathsf{Verify}(pk, m, \sigma)$

Output 1, iff $\sigma_1 \neq 1_{\mathbb{G}_1}$ and $\mathrm{e}(\sigma_1, \tilde{X} \cdot \tilde{Y}^m) = \mathrm{e}(\sigma_2, \tilde{g})$.

To sign committed messages $M = g_1^m$ a modified signing algorithm can be used:

$\mathsf{Sign}(sk, M)$

Choose $\alpha \leftarrow \mathbb{Z}_p$, set $\sigma = (\sigma_1, \sigma_2) := (g_1^\alpha, (g_1^{\xi_1} \cdot M^{\xi_2})^\alpha)$, and output σ as signature on m.

Definition 3.5 (Trapdoor Pedersen Commitments (PD))**.** Let $\mathbb{GD} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathrm{e}, g_1, g_2)$ be a bilinear group setting of Type-3, with generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. The Trapdoor Pedersen Commitment Scheme for messages $m \in \mathbb{Z}_p$ is defined as follows:

$\mathsf{KeyGen}(\mathbb{GD})$

Choose $td \xleftarrow{u} \mathbb{Z}_p$, $u \xleftarrow{u} \mathbb{G}_1$, set $v := u^{td}$, and output $pk := (u, v)$.

$\mathsf{Commit}(pk, m)$

Choose $r \xleftarrow{u} \mathbb{Z}_p$ and output the commitment $com := u^m \cdot v^r$ and the decommitment r.

$\mathsf{Reveal}(pk, com, m, r)$

Output the decommitment r such that $\mathsf{Commit}(pk, m; r) = com$.

$\mathsf{Equiv}(pk, td, com, m, r, m')$

Output $r' := (m - m' + td \cdot r) \cdot td^{-1}$.

To rate a product a user has to non-interactively prove knowledge of the registration token, the rating token, and its personal user-secret, for which the tokens were generated. As non-interactive proof system we use Signatures of Knowledge [CL06]. Also, *opening-proofs*, generated by P_{IDM}, are non-interactive proofs of knowledge of *opening tokens*. These tokens are given by a user P_i to the System Manager P_{IDM} during the registration protocol. In our construction it is important not to publish these tokens, because they allow to open any rating. Hence, we encrypt opening tokens with the CCA2-secure *Cramer-Shoup encryption* ($\mathsf{CS} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$) [CS98].

Definition 3.6 (Cramer-Shoup Encryption (CS))**.** Let $\mathbb{GD} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathrm{e}, g_1, g_2)$ be a bilinear group setting of Type-3, with generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, and let $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p$ be a collision resistant hash function. The Cramer-Shoup Encryption Scheme for messages $m \in \mathbb{G}_2$ is defined as follows:

$\mathsf{KeyGen}(\mathbb{GD})$

1: Choose $\tilde{h} \xleftarrow{u} \mathbb{G}_2$ and $\zeta_1, \zeta_2, \zeta_3, \zeta_4, \zeta_5 \xleftarrow{u} \mathbb{Z}_p$.

2: Set $\tilde{b} := g_2^{\zeta_1} \cdot \tilde{h}^{\zeta_2}, \tilde{d} := g_2^{\zeta_3} \cdot \tilde{h}^{\zeta_4}, \tilde{f} := g_2^{\zeta_5}$, $sk := (\zeta_1, \zeta_2, \zeta_3, \zeta_4, \zeta_5)$, and $pk := (g_2, \tilde{h}, \tilde{b}, \tilde{d}, \tilde{f}, \mathcal{H})$.

3: Output the key pair (sk, pk).

$\mathsf{Enc}(pk, m)$

1: Choose $\beta \xleftarrow{u} \mathbb{Z}_p$

2: Set $ct_1 := g_2^{\beta}$, $ct_2 := \tilde{h}^{\beta}$, $ct_3 := m \cdot \tilde{f}^{\beta}$, $\omega := \mathcal{H}(ct_1, ct_2, ct_3)$, $ct_4 := (\tilde{b} \cdot \tilde{d}^{\omega})^{\beta}$.

3: Output the cipher text $ct := (ct_1, ct_2, ct_3, ct_4)$.

$\mathsf{Dec}(sk, ct)$

Output $m := ct_3 \cdot ct_1^{-\zeta_5}$, iff $ct_1^{\zeta_1} \cdot ct_2^{\zeta_2} \cdot (ct_1^{\zeta_3} \cdot ct_2^{\zeta_4})^{\omega} = ct_4$, where $\omega := \mathcal{H}(ct_1, ct_2, ct_3)$.

The Signatures of Knowledge we use need a Random Oracle, which can be modeled as the ideal functionality $\mathcal{F}_{\mathsf{RO}}$ [HMQ04] in the UC framework. We further need the ideal functionalities for *Common Reference Strings* $\mathcal{F}_{\mathsf{CRS}}$ [CF01] and *Certification* $\mathcal{F}_{\mathsf{CA}}$ [Can04]. $\mathcal{F}_{\mathsf{CRS}}$ is needed for secure commitment schemes like the above mentioned Trapdoor Pedersen Commitments and $\mathcal{F}_{\mathsf{CA}}$ ensures that users cannot register with different identities.

$\mathcal{F}_{\mathsf{RO}}$

$\mathcal{F}_{\mathsf{RO}}$ operates on security parameter k and manages the list L_{RO} of pairs of bitstrings, which is initially empty.

Retrieving values: On input (sid, m) from P or $\mathcal{S}$

```
If $(m, v) \in L_{RO}$ for some $v \in \{0,1\}^k$ then set $h := v$
Else choose $h \xleftarrow{\$} \{0,1\}^k$ and store $(m, h)$ in $L_{RO}$
Send $(sid, m, h)$ to the activating party ($P$ or $\mathcal{S}$).
```

Functionality 2: Random Oracle

$\mathcal{F}_{\mathsf{CRS}}$

$\mathcal{F}_{\mathsf{CRS}}$ operates on distribution D and controls the value CRS, which is initialized to $\perp$.

Retrieving the CRS: On input (sid) from P or $\mathcal{S}$

```
If $CRS = \perp$ then set $CRS \leftarrow D$.
Send $(sid, CRS)$ to the activating party ($P$ or $\mathcal{S}$).
```

Functionality 3: Common Reference String

$\mathcal{F}_{\mathsf{CA}}$

Registering Values: On input $(\mathsf{Register}, sid, v)$ from P_i

```
Send $(\mathsf{Register}, sid)$ to $\mathcal{S}$ and receive $(\mathsf{Register}, sid, \mathtt{ok})$ from $\mathcal{S}$.
If $sid = P_i$ and this is the first request then record $(P_i, v)$.
```

Retrieving registered values: On input $(\mathsf{Retrieve}, sid)$ from P_j

```
Send $(\mathsf{Retrieve}, sid, P_j)$ to $\mathcal{S}$ and receive $(\mathsf{Retrieve}, sid, P_j, \mathtt{ok})$ from $\mathcal{S}$.
If there is a tuple $(sid, v)$ recorded then send $(\mathsf{Retrieve}, sid, v)$ to $P_j$.
Else send $(\mathsf{Retrieve}, sid, \perp)$ to $P_j$.
```

Functionality 4: Certification Authority

In our construction the output of $\mathcal{F}_{\mathsf{CRS}}$ is $(\mathbb{GD}, \mathsf{PD}.pk, \mathcal{H}, \mathcal{H}_1, \mathcal{H}_2)$, where $\mathbb{GD}$ is the output of the bilinear group generator $\mathsf{BiGrGen}(1^\lambda)$, $\mathsf{PD}.pk = (u, v) \in \mathbb{G}_1^2$ is the public key of the Trapdoor Pedersen Commitment scheme, and $\mathcal{H} \colon \{0,1\}^* \to \mathbb{Z}_p$, $\mathcal{H}_1 \colon \{0,1\}^* \to \mathbb{G}_1$, and $\mathcal{H}_2 \colon \{0,1\}^* \to \mathbb{G}_2$ are collision-resistant hash functions. We assume that every party obtains the common-reference string prior to its first activation.

We write $y := \mathcal{F}_{\mathsf{RO}}(x)$ to indicate a call to $\mathcal{F}_{\mathsf{RO}}$ on input (sid, x) and outputting y to the calling party.

3.3 A Protocol for Realizing $\mathcal{F}_{\mathsf{RS}}$

We assume to communicate via authenticated channels between two parties. This implies that the identities of communicating parties are known to each other and that the adversary cannot modify the message's payload.

Π_{RS}

All parties except P_{IDM}: On the first activation of P_i

```
1: Choose a value $usk_i \overset{\$}{\leftarrow} \mathbb{Z}_p$ and compute $M_i := g_1^{usk_i}$, where $g_1 \in \mathbb{G}_1$ is given by $\mathcal{F}_{\mathsf{CRS}}$.
2: Send $(\mathsf{Register}, P_i, M_i)$ to $\mathcal{F}_{\mathsf{CA}}$, and store the user-secret-key $usk_i$.
```

Registry Key Generation: When P_{IDM} receives (KeyGen, sid) from $\mathcal{Z}$

```
1: Run $\mathsf{PS.KeyGen}(\mathbb{GD})$ to obtain $\mathsf{PS}.pk := (\tilde{g}, \tilde{X}, \tilde{Y})$ and $\mathsf{PS}.sk := (\xi_1, \xi_2)$.
2: Run $\mathsf{CS.Setup}(\mathbb{GD})$ to obtain $\mathsf{CS}.pk := (g_2, \tilde{h}, \tilde{b}, \tilde{d}, \tilde{f}, \mathcal{H})$ and $\mathsf{CS}.sk := (\zeta_1, \zeta_2, \zeta_3, \zeta_4, \zeta_5)$.
3: Set $pp := (\mathsf{PS}.pk, \mathsf{CS}.pk)$ and $idmsk := (\mathsf{PS}.sk, \mathsf{CS}.sk)$.
4: Set $\mathfrak{Params}.\mathsf{Add}(pp)$ and $\mathfrak{Params}_s.\mathsf{Add}(pp, idmsk)$.
5: Output $(\mathsf{KeyGen}, sid, pp)$.
```

User Registration: When P_i receives $(\mathsf{Register}, sid, pp')$ from $\mathcal{Z}$

```
$P_i$:       1: Choose $\alpha, r \overset{\$}{\leftarrow} \mathbb{Z}_p$, compute $T := g_1^{\alpha}$, $R := u^{\mathcal{H}(T)} \cdot v^r$ and send $(pp', R)$ to $P_{\mathrm{IDM}}$.
$P_{\mathrm{IDM}}$:  2: Obtain $M_i$ from $\mathcal{F}_{\mathsf{CA}}(\mathsf{Retrieve}, P_i)$.
             3: If $\mathcal{F}_{\mathsf{CA}}$ returned $(\mathsf{Retrieve}, P_i, \perp)$, $pp' \notin \mathfrak{Params}$ or $(P_i, pp', M', Y', \sigma') \in \mathfrak{Reg}$ for some
                   $M'$, $Y'$, $\sigma'$ then send abort to $P_i$ and output $(\mathsf{Register}, sid, pp', P_i, 0)$.
             4: Else Choose $ch \overset{\$}{\leftarrow} \mathbb{Z}_p$ and send $ch$ to $P_i$.
$P_i$:       5: If $P_{\mathrm{IDM}}$ sent abort then output $(\mathsf{Register}, sid, pp', P_i, 0)$.
             6: Else Compute $s_\alpha := \alpha + ch \cdot usk_i$, $ct \leftarrow \mathsf{CS.Enc}(\mathsf{CS}.pk, \tilde{Y}^{usk_i})$ and send $(s_\alpha, T, r, ct)$
                   to $P_{\mathrm{IDM}}$.
$P_{\mathrm{IDM}}$:  7: Compute $\tilde{Y}_i := \mathsf{CS.Dec}(\mathsf{CS}.sk, ct)$.
             8: If decrypting $ct$ failed, or $M_i^{ch} \cdot T \neq g_1^{s_\alpha}$, or $R \neq u^{\mathcal{H}(T)} \cdot v^r$, or $\mathrm{e}(M_i, \tilde{Y}) \neq \mathrm{e}(g_1, \tilde{Y}_i)$
                   then send abort to $P_i$ and output $(\mathsf{Register}, sid, pp', P_i, 0)$.
             9: Else compute $\sigma_i \leftarrow \mathsf{PS.Sign}(\mathsf{PS}.sk, M_i)$, set $\mathfrak{Reg}.\mathsf{Add}(P_i, pp', M_i, \tilde{Y}_i, \sigma_i)$, send $\sigma_i$ to $P_i$,
                   and output $(\mathsf{Register}, sid, pp', P_i, 1)$.
$P_i$:      10: If $P_{\mathrm{IDM}}$ sent abort then output $(\mathsf{Register}, sid, pp', P_i, 0)$.
            11: Else If $\mathsf{PS.Verify}(pp', usk_i, \sigma_i) = 1$ then
            12:         store $(usk_i, \sigma_i)$, and output $(\mathsf{Register}, sid, pp', P_i, 1)$.
            13: Else output $(\mathsf{Register}, sid, pp', P_i, 0)$.
```

Product Addition: When P_i receives $(\mathsf{NewProduct}, sid, prod)$ from $\mathcal{Z}$

```
1: Compute $\tilde{g}_{i,prod} := \mathcal{H}_2(i, prod)$ and run $\mathsf{PS.KeyGen}(\mathbb{GD})$ with $\tilde{g}_{i,prod}$ as generator of $\mathbb{G}_2$ to
      obtain $\mathsf{PS}.pk_{i,prod} := (\tilde{g}_{i,prod}, \tilde{X}_{i,prod}, \tilde{Y}_{i,prod})$ and $\mathsf{PS}.sk_{i,prod} := (\xi_{1_{i,prod}}, \xi_{2_{i,prod}})$.
```

```
Compute $M_{i,prod} := \mathcal{H}_1(i, prod)^{usk_i}$.
Choose $r \overset{\$}{\leftarrow} \mathbb{Z}_p$ and compute $R_1 := \mathcal{H}_1(i, prod)^r$ and $R_2 := g_1^r$.
Set $ch_{i,prod} := \mathcal{F}_{\mathsf{RO}}(\mathsf{PS}.pk_{i,prod}, M_i, M_{i,prod}, R_1, R_2)$ and $s_{i,prod} := r + ch_{i,prod} \cdot usk_i$.
Set $ppk_{i,prod} := (M_i, M_{i,prod}, ch_{i,prod}, s_{i,prod}, \mathsf{PS}.pk_{i,prod})$ and $\mathfrak{Prods}_i.\mathsf{Add}(prod, ppk_{i,prod})$.
Output $(\mathsf{NewProduct}, sid, prod, ppk_{i,prod})$.
```

Purchase: When P_i receives $(\mathsf{Purchase}, sid, P_j, prod, ppk)$ from $\mathcal{Z}$

```
$P_i$: If $\mathsf{VfyProd}(P_j, prod, ppk) = 0 \lor P_i = P_j$ then ignore the request.
       Else choose $\alpha, r \overset{\$}{\leftarrow} \mathbb{Z}_p$, compute $T := g_1^{\alpha}$, $R := u^{\mathcal{H}(T)} \cdot v^r$ and send $(prod, ppk, R)$ to $P_j$.
$P_j$: Obtain $M_i$ from $\mathcal{F}_{\mathsf{CA}}(\mathsf{Retrieve}, P_i)$.
       If $\mathcal{F}_{\mathsf{CA}}$ returned $(\mathsf{Retrieve}, P_i, \perp)$ or $(prod, ppk) \notin \mathfrak{Prods}_j$ then
           send abort to $P_i$ and output $(\mathsf{Purchase}, sid, P_i, P_j, prod, ppk, 0)$.
       Else choose $ch \overset{\$}{\leftarrow} \mathbb{Z}_p$ and send $ch$ to $P_i$.
$P_i$: If $P_j$ sent abort then output $(\mathsf{Purchase}, sid, P_i, P_j, prod, ppk, 0)$.
       Else compute $s_\alpha := \alpha + ch \cdot usk_i$ and send $(s_\alpha, T, r)$ to $P_j$.
$P_j$: If $M_i^{ch} \cdot T \neq g_1^{s_\alpha}$ or $R \neq u^{\mathcal{H}(T)} \cdot v^r$ then
           send abort to $P_i$ and output $(\mathsf{Purchase}, sid, P_i, P_j, prod, ppk, 0)$.
       Else compute $\sigma_{i,j,prod} \leftarrow \mathsf{PS.Sign}(\mathsf{PS}.sk_{i,prod}, M_i)$ and set $\mathfrak{Purch}_j.\mathsf{Add}(P_i, prod, \sigma_{i,j,prod})$.
               Send $\sigma_{i,j,prod}$ to $P_i$.
$P_i$: If $P_j$ sent abort then output $(\mathsf{Purchase}, sid, P_i, P_j, prod, ppk, 0)$.
       Else If $\mathsf{PS.Verify}(\mathsf{PS}.pk_{i,prod}, usk_i, \sigma_{i,j,prod}) = 1$ then
                   store $\sigma_{i,j,prod}$, and output $(\mathsf{Purchase}, sid, P_i, P_j, prod, ppk, 1)$.
       Else output $(\mathsf{Purchase}, sid, P_i, P_j, prod, ppk, 0)$.
```

$\mathsf{VfyProd}$: On local input $(P_j, prod, ppk)$

```
Obtain $M_j$ from $\mathcal{F}_{\mathsf{CA}}(\mathsf{Retrieve}, P_j)$
If $\mathcal{F}_{\mathsf{CA}}$ returned $(\mathsf{Retrieve}, P_j, \perp)$ then return 0.
Else parse $ppk$ as $(M_j', M_{j,prod}, ch_{j,prod}, s_{j,prod}, \mathsf{PS}.pk_{j,prod})$.
        Set $R_1 := \mathcal{H}_1(j, prod)^{s_{j,prod}} \cdot M_{j,prod}^{-ch_{j,prod}}$ and $R_2 := g_1^{s_{j,prod}} \cdot M_j^{-ch_{j,prod}}$.
If $M_j \neq M_j'$ or $ch_{j,prod} \neq \mathcal{F}_{\mathsf{RO}}(\mathsf{PS}.pk_{j,prod}, M_j, M_{j,prod}, R_1, R_2)$ then return 0.
Else return 1.
```

Rate a Product: When P_i receives $(\mathsf{Rate}, sid, pp, P_j, prod, ppk, m)$ from $\mathcal{Z}$

```
If no tuple $(usk_i, \sigma_i)$ is stored such that $\mathsf{PS.Verify}(pp, usk_i, \sigma_i) = 1$, or no $\sigma_{i,j,prod}$ is stored
    such that $\mathsf{PS.Verify}(\mathsf{PS}.pk_{i,prod}, usk_i, \sigma_{i,j,prod}) = 1$, or a tuple $(m', \sigma) = (m', T_1, T_2, T_3,$
    $T_4, T_5, ch, s)$ is stored such that $(\mathsf{Verify}, sid, pp, P_j, prod, ppk, m', \sigma) = 1$ then
    ignore the request.
Choose $t_1, t_2, k \overset{\$}{\leftarrow} \mathbb{Z}_p$.
Compute $T_1 := \sigma_{i,1}^{t_1}$, $T_2 := \sigma_{i,2}^{t_1}$, $T_3 := \sigma_{i,j,prod,1}^{t_2}$, $T_4 := \sigma_{i,j,prod,2}^{t_2}$, $T_5 := \mathcal{H}_1(j, prod)^{usk_i}$.
Compute $R_1 := \mathrm{e}(T_1, \tilde{Y})^k$, $R_2 := \mathrm{e}(T_3, \tilde{Y}_{j,prod})^k$, $R_3 := \mathcal{H}_1(j, prod)^k$.
Set $ch := \mathcal{F}_{\mathsf{RO}}(T_1, T_2, T_3, T_4, T_5, R_1, R_2, R_3, prod, ppk, m)$, and $s := k + ch \cdot usk_i$.
Set $\sigma := (T_1, T_2, T_3, T_4, T_5, ch, s)$ and store $\sigma$.
Output $(\mathsf{Rate}, sid, pp, P_j, prod, ppk, m, \sigma)$.
```

Verifying a Rating: When P_i receives $(\mathsf{Verify}, sid, pp, P_j, prod, ppk, m, \sigma)$ from $\mathcal{Z}$

```
If $\mathsf{VfyProd}(P_j, prod, ppk) = 0$ then ignore the request.
Parse $\sigma$ as $(T_1, T_2, T_3, T_4, T_5, ch, s)$.
Set $R'_1 := \mathrm{e}(T_1, \tilde{X})^{ch} \cdot \mathrm{e}(T_2, \tilde{g})^{-ch} \cdot \mathrm{e}(T_1, \tilde{Y})^s$, $R'_3 := T_5^{-ch} \cdot \mathcal{H}_1(j, prod)^s$,
    $R'_2 := \mathrm{e}(T_3, \tilde{X}_{j,prod})^{ch} \cdot \mathrm{e}(T_4, \tilde{g}_{j,prod})^{-ch} \cdot \mathrm{e}(T_3, \tilde{Y}_{j,prod})^s$.
Set $f := \big[T_5 \neq M_{j,prod} \wedge ch = \mathcal{F}_{\mathsf{RO}}(T_1,\ T_2,\ T_3,\ T_4,\ T_5,\ R'_1,\ R'_2,\ R'_3,\ prod,\ ppk,\ m)\big]$
Output $(\mathsf{Verify}, sid, pp, P_j, prod, ppk, m, \sigma, f)$.
```

Linking Ratings: When P_i receives $(\mathsf{Link}, sid, pp, P_j, prod, ppk, m_0, \sigma_0, m_1, \sigma_1)$ from $\mathcal{Z}$

```
If $(\mathsf{Verify}, sid, pp, P_j, prod, ppk, m_k, \sigma_k) = 1$, for $k \in \{0, 1\}$ then
    Parse $\sigma_0$ as $(T_1, T_2, T_3, T_4, T_5, ch, s)$, and $\sigma_1$ as $(T'_1, T'_2, T'_3, T'_4, T'_5, ch', s')$.
    Output $(\mathsf{Link}, sid, pp, P_j, prod, ppk, m_0, \sigma_0, m_1, \sigma_1, (T_5 = T'_5))$.
Else Output $(\mathsf{Link}, sid, pp, P_j, prod, ppk, m_0, \sigma_0, m_1, \sigma_1, 0)$.
```

Determine Raters Identity: When P_{IDM} receives $(\mathsf{Open}, sid, pp, P_j, prod, ppk, m, \sigma)$ from $\mathcal{Z}$

```
If $pp \notin \mathfrak{Params}$ then ignore the request.
Set $f := (\mathsf{Verify}, sid, pp, P_j, prod, ppk, m, \sigma)$.
If $f = 1$ then parse $\sigma$ as $(T_1, T_2, T_3, T_4, T_5, ch, s)$ and iterate through $\mathfrak{Reg}$ to find a tuple
        $(P_i, pp, M_i, \tilde{Y}_i, \sigma_i)$ such that $\mathrm{e}(T_5, \tilde{Y}) = \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{Y}_i)$.
If $f = 0$ or no such tuple could be found then output $(\mathsf{Open}, sid, pp, P_j, prod, ppk, m, \sigma, \perp)$.
Else Set $\mathfrak{Open}.\mathsf{Add}(pp, P_j, prod, ppk, m, \sigma, P_i)$.
        Output $(\mathsf{Open}, sid, pp, P_j, prod, ppk, m, \sigma, P_i)$.
```

Generate Opening Proof: When P_{IDM} receives $(\mathsf{OProof}, sid, pp, P_j, prod, ppk, m, \sigma, P)$ from $\mathcal{Z}$

```
If $pp \notin \mathfrak{Params}$ then ignore the request.
Set $f := (\mathsf{Verify}, sid, pp, P_j, prod, ppk, m, \sigma)$.
If $f = 0 \vee (pp, P_j, prod, ppk, m, \sigma, P) \notin \mathfrak{Open}$ then
    output $(\mathsf{OProof}, sid, pp, P_j, prod, ppk, m, \sigma, P, \perp)$.
Else Parse $\sigma$ as $(T_1, T_2, T_3, T_4, T_5, ch, s)$.
        Select the tuple $(P, pp, M_i, \tilde{Y}_i, \sigma_i)$ such that $\mathrm{e}(T_5, \tilde{Y}) = \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{Y}_i)$.
        Choose $\beta \xleftarrow{\$} \mathbb{Z}_p$ and compute $ct = (ct_1, ct_2, ct_3, ct_4) \leftarrow \mathsf{CS.Enc}(\mathsf{CS}.pk, \tilde{Y}_i; \beta)$.
        Choose $r \xleftarrow{\$} \mathbb{Z}_p$ and compute $R_1 := g_2^r$, $R_2 := \tilde{h}^r$, $R_3 := \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{f})^r$.
        Compute $\omega := \mathcal{H}(ct_1, ct_2, ct_3)$, $R_4 := (\tilde{b} \cdot \tilde{d}^{\omega})^r$, $R_5 := \mathrm{e}(g_1, \tilde{f})^r$.
        Set $\hat{ch} := \mathcal{F}_{\mathsf{RO}}(ct, R_1, R_2, R_3, R_4, R_5, \sigma, i, M_i)$, $\hat{s} := r + \hat{ch} \cdot \beta$, and $\tau := (P_i, ct, \hat{ch}, \hat{s})$.
        Set $\mathfrak{Open}.\mathsf{Add}(pp, P_j, prod, ppk, m, \sigma, P, \tau)$.
        Output $(\mathsf{OProof}, sid, pp, P_j, prod, ppk, m, \sigma, P, \tau)$.
```

Verifying Opening-Proofs: When P_i receives $(\mathsf{Judge}, sid, pp, P_j, prod, ppk, m, \sigma, P, \tau)$ from $\mathcal{Z}$

```
Set $f := (\mathsf{Verify}, sid, pp, P_j, prod, ppk, m, \sigma)$.
Obtain $M$ from $\mathcal{F}_{\mathsf{CA}}(\mathsf{Retrieve}, P)$.
**If** $\mathcal{F}_{\mathsf{CA}}$ returned $(\mathsf{Retrieve}, P, \perp) \vee f = 0 \vee P = \perp \vee \tau = \perp$ **then**
    output $(\mathsf{Judge}, sid, pp, P_j, prod, ppk, m, \sigma, P, \tau, 0)$.
**Else** Parse $\tau$ as $(P_i, (ct_1, ct_2, ct_3, ct_4), \hat{ch}, \hat{s})$.
        Compute $R_1 := ct_1^{-\hat{ch}} \cdot g_2^{\hat{s}}$, $R_2 := ct_2^{-\hat{ch}} \cdot \tilde{h}^{\hat{s}}$.
        Compute $R_3 := \mathrm{e}(\mathcal{H}_1(j, prod), ct_3)^{-\hat{ch}} \cdot \mathrm{e}(T_5, \tilde{Y})^{\hat{ch}} \cdot \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{f})^{\hat{s}}$
        Compute $\omega := \mathcal{H}(ct_1, ct_2, ct_3)$.
        Compute $R_4 := ct_4^{-\hat{ch}} \cdot (\tilde{b} \cdot \tilde{d}^{\omega})^{\hat{s}}$, $R_5 := \mathrm{e}(g_1, ct_3)^{-\hat{ch}} \cdot \mathrm{e}(M, \tilde{Y})^{\hat{ch}} \cdot \mathrm{e}(g_1, \tilde{f})^{\hat{s}}$.
        Set $f := (P = P_i \wedge \hat{ch} = \mathcal{F}_{\mathsf{RO}}(ct, R_1, R_2, R_3, R_4, R_5, \sigma, i, M))$.
        Output $(\mathsf{Judge}, sid, pp, P_j, prod, ppk, m, \sigma, P, \tau, f)$
```

Protocol 1: Protocol for $\mathcal{F}_{\mathsf{RS}}$

Theorem 3.1. *Under the* Authenticated Channels Assumption*, the* SXDH-Assumption*, the* Pointcheval-Sanders-Assumption*, and the assumption that $\mathcal{H}, \mathcal{H}_1$, and $\mathcal{H}_2$ are collision-resistant hash functions, Protocol Π_{RS} UC-realizes the $\mathcal{F}_{\mathsf{RS}}$ functionality in the ($\mathcal{F}_{\mathsf{RO}}$, $\mathcal{F}_{\mathsf{CRS}}$, $\mathcal{F}_{\mathsf{CA}}$)-hybrid model, in the presence of static adversaries.*

3.3.1 Intuition to the Proof

To prove Theorem 3.1 we have to show that for any probabilistic polynomial-time real-world adversary $\mathcal{A}$ there exists a probabilistic polynomial-time ideal-world adversary $\mathcal{S}$ such that for any probabilistic polynomial-time environment $\mathcal{Z}$ it holds:

$$\left\{\mathrm{EXEC}_{\mathcal{F}_{\mathsf{RS}},\mathcal{S}^{\mathcal{A}},\mathcal{Z}}(1^\lambda, z)\right\}_{\lambda\in\mathbb{N}, z\in\{0,1\}^*} \overset{c}{\equiv} \left\{\mathrm{EXEC}^{\mathcal{F}_{\mathsf{RO}},\mathcal{F}_{\mathsf{CRS}},\mathcal{F}_{\mathsf{CA}}}_{\Pi_{\mathsf{RS}},\mathcal{A},\mathcal{Z}}(1^\lambda, z)\right\}_{\lambda\in\mathbb{N}, z\in\{0,1\}^*} .$$

We divide the proof of this statement into three parts. In the first part we define the simulator $\mathcal{S}$ that interacts with $\mathcal{F}_{\mathsf{RS}}$ and simulates the cryptographic computations. Note that during Rate-requests $\mathcal{S}$ does not obtain any identifying information of the rater. Hence, $\mathcal{S}$ uses the zero-knowledge simulator for the Signature of Knowledge that represents a rating. Analogously, opening-proofs are represented by a Signature of Knowledge. Therefore, $\mathcal{S}$ uses the corresponding zero-knowledge simulator to generate opening-proofs.

In the second part of the proof we define a hybrid game $\mathcal{G}$ and a corresponding simulator $\mathcal{S}_1$ for which we prove that no environment $\mathcal{Z}$ can distinguish whether it interacts with $(\mathcal{F}_{\mathsf{RS}}, \mathcal{S})$ or $(\mathcal{G}, \mathcal{S}_1)$. In this game $\mathcal{S}_1$ obtains all identifying information during Rate-requests and therefore can execute the computations as defined in Protocol Π_{RS}. Also opening-proofs can be generated by $\mathcal{S}_1$ as in Protocol Π_{RS}. Hence, an environment $\mathcal{Z}$ is only able to distinguish $(\mathcal{F}_{\mathsf{RS}}, \mathcal{S})$ and

$(\mathcal{G}, \mathcal{S}_1)$, if it can distinguish between simulated and real ratings and opening-proofs. Under the *SXDH-Assumption* this is not possible.

In the third part of the proof we show that $\mathcal{S}_1$ executes exactly the same computations as Protocol Π_{RS}. This implies that any environment $\mathcal{Z}$ that distinguishes between $(\mathcal{G}, \mathcal{S}_1)$ and $(\Pi_{\mathsf{RS}}, \mathcal{A})$ is able to let $\mathcal{F}_{\mathsf{RS}}$ output `error`, whereas the Protocol Π_{RS} outputs some value, or $\mathcal{F}_{\mathsf{RS}}$ outputs 0, whereas Protocol Π_{RS} outputs 1 (or vice versa). Using different reductions to the Pointcheval-Sanders-Problem and to the CCA2-security of the Cramer-Shoup encryption scheme we show that such environments cannot exist. Hence, Π_{RS} UC-realizes $\mathcal{F}_{\mathsf{RS}}$ in the ($\mathcal{F}_{\mathsf{RO}}$, $\mathcal{F}_{\mathsf{CRS}}$, $\mathcal{F}_{\mathsf{CA}}$)-hybrid model.

3.4 The Proof of Theorem 3.1

Formally, we prove the following: for every $\mathcal{A}$ and every $\mathcal{Z}$

$$\left\{\mathrm{EXEC}_{\mathcal{F}_{\mathsf{RS}}, \mathcal{S}^{\mathcal{A}}, \mathcal{Z}}(1^\lambda, z)\right\}_{\lambda \in \mathbb{N}, z \in \{0,1\}^*} \overset{c}{\equiv} \left\{\mathrm{EXEC}^{\mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{CA}}}_{\Pi_{\mathsf{RS}}, \mathcal{A}, \mathcal{Z}}(1^\lambda, z)\right\}_{\lambda \in \mathbb{N}, z \in \{0,1\}^*}$$

by introducing a hybrid game $\mathcal{G}$ and proving the two relations

$$\left\{\mathrm{EXEC}_{\mathcal{F}_{\mathsf{RS}}, \mathcal{S}^{\mathcal{A}}, \mathcal{Z}}(1^\lambda, z)\right\}_{\lambda \in \mathbb{N}, z \in \{0,1\}^*} \overset{c}{\equiv} \left\{\mathcal{G}_{\mathcal{F}_{\mathsf{RS}}, \mathcal{S}_1^{\mathcal{A}}, \mathcal{Z}}(1^\lambda, z)\right\}_{\lambda \in \mathbb{N}, z \in \{0,1\}^*}$$

and

$$\left\{\mathcal{G}_{\mathcal{F}_{\mathsf{RS}}, \mathcal{S}_1^{\mathcal{A}}, \mathcal{Z}}(1^\lambda, z)\right\}_{\lambda \in \mathbb{N}, z \in \{0,1\}^*} \overset{c}{\equiv} \left\{\mathrm{EXEC}^{\mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{CA}}}_{\Pi_{\mathsf{RS}}, \mathcal{A}, \mathcal{Z}}(1^\lambda, z)\right\}_{\lambda \in \mathbb{N}, z \in \{0,1\}^*} .$$

As abbreviations we set

$$\mathrm{IDEAL} := \left\{\mathrm{EXEC}_{\mathcal{F}_{\mathsf{RS}}, \mathcal{S}^{\mathcal{A}}, \mathcal{Z}}(1^\lambda, z)\right\}_{\lambda \in \mathbb{N}, z \in \{0,1\}^*}$$

and

$$\mathrm{HYBRID} := \left\{\mathrm{EXEC}^{\mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{CA}}}_{\Pi_{\mathsf{RS}}, \mathcal{A}, \mathcal{Z}}(1^\lambda, z)\right\}_{\lambda \in \mathbb{N}, z \in \{0,1\}^*} .$$

3.4.1 Foundations for the Proof of Theorem 3.1

The protocols $\Pi_{\mathsf{RS}}.\mathsf{Register}$, $\Pi_{\mathsf{RS}}.\mathsf{NewProduct}$, $\Pi_{\mathsf{RS}}.\mathsf{Purchase}$, $\Pi_{\mathsf{RS}}.\mathsf{Rate}$, and $\Pi_{\mathsf{RS}}.\mathsf{OProof}$ are based on zero-knowledge proofs of knowledge. In this section we prove these properties because they are used in the proof of Theorem 3.1. To formalize the statements to prove we use the notation introduced by Camenisch and Stadler [CS97].

Lemma 3.1. *In the protocols* $\Pi_{\mathsf{RS}}.\mathsf{Register}$ *and* $\Pi_{\mathsf{RS}}.\mathsf{Purchase}$ *party* P_i *proves the statement* $ZKPK\{(usk_i) : M_i = g^{usk_i}\}$ *to* P_{IDM} *and* P_j*, respectively.*

Proof.

- Completeness: An honest prover will generate an accepting transcript as the verification equations hold: $M_i^{ch} \cdot T = (g_1^{usk_i})^{ch} \cdot g_1^{\alpha} = g_1^{\alpha + ch \cdot usk_i} = g_1^{s_\alpha}$ and $R = u^{\mathcal{H}(T)} \cdot v^r$.

- Interactive Simulator: In order to simulate transcripts of the protocol, the simulator has to set up the Trapdoor Pedersen Commitment (Definition 3.5). By running the KeyGen algorithm the simulator knows the trapdoor td. With this value the simulation works as follows:

  Choose $\alpha', r' \xleftarrow[u]{} \mathbb{Z}_p$, compute $T' := g_1^{\alpha'}$, $R := u^{\mathcal{H}(T')} \cdot v^{r'}$ and send the commitment R to the verifier. On receiving a challenge $ch \in \mathbb{Z}_p$ the simulator chooses $s_\alpha \xleftarrow[u]{} \mathbb{Z}_p$ and sets $T := g_1^{s_\alpha} \cdot M_i^{-ch}$. Finally, by using the commitment trapdoor td the simulator computes $r := (\mathcal{H}(T') - \mathcal{H}(T) + td \cdot r') \cdot td^{-1}$, according to the Equiv algorithm, and outputs (s_α, T, r) to the verifier. The resulting transcripts are identically distributed as real transcripts.

- Extractor: Given two accepting transcripts (R, ch, s_α, T, r) and $(R, ch', s'_\alpha, T, r)$ the extractor computes $usk_i := (s_\alpha - s'_\alpha)/(ch - ch')$, which is the discrete logarithm of M_i to base g_1: $M_i^{ch} \cdot T = g_1^{s_\alpha} \wedge M_i^{ch'} \cdot T = g_1^{s'_\alpha} \Longrightarrow M_i^{ch-ch'} = g_1^{s_\alpha - s'_\alpha} \Longrightarrow M_i = g_1^{(s_\alpha - s'_\alpha)/(ch-ch')}$. □

Lemma 3.2. *The value $ppk_{i,prod}$ output in Protocol $\Pi_{\mathsf{RS}}.\mathsf{NewProduct}$ is a Signature of Knowledge on message $\mathsf{PS}.pk_{i,prod}$ proving the statement $ZKPK\{(usk_i) : M_i = g_1^{usk_i} \wedge M_{i,prod} = \mathcal{H}_1(i, prod)^{usk_i}\}$.*

Proof.

- Completeness: An honest prover will generate an accepting transcript as the verification equations hold:

$$\begin{aligned} \mathcal{H}_1(i, prod)^{s_{i,prod}} \cdot M_{i,prod}^{-ch_{i,prod}} &= \mathcal{H}_1(i, prod)^{r + ch_{i,prod} \cdot usk_i} \cdot \mathcal{H}_1(i, prod)^{-usk_i \cdot ch_{i,prod}} \\ &= \mathcal{H}_1(i, prod)^r = R_1 && (1) \\ g_1^{s_{i,prod}} \cdot M_i^{-ch_{i,prod}} &= g_1^{r + ch_{i,prod} \cdot usk_i} \cdot g_1^{-usk_i \cdot ch_{i,prod}} = g_1^r = R_2 && (2) \end{aligned}$$

  and hence

$$ch_{i,prod} = \mathcal{F}_{\mathsf{RO}}(\mathsf{PS}.pk_{i,prod}, M_i, M_{i,prod}, R_1, R_2). \qquad (3)$$

- Simulator: Given $M_i, M_{i,prod}$, and $\mathsf{PS}.pk_{i,prod}$ as input and using the random oracle $\mathcal{F}_{\mathsf{RO}}$, transcripts can be simulated, as follows:

  Choose $ch_{i,prod}, s_{i,prod} \xleftarrow[u]{} \mathbb{Z}_p$, set $R_1 := \mathcal{H}_1(i, prod)^{s_{i,prod}} \cdot M_{i,prod}^{-ch_{i,prod}}$ and $R_2 := g_1^{s_{i,prod}} \cdot M_i^{-ch_{i,prod}}$, and patch $\mathcal{F}_{\mathsf{RO}}(\mathsf{PS}.pk_{i,prod}, M_i, M_{i,prod}, R_1, R_2) := c_{i,prod}$. The resulting transcripts are identically distributed as real transcripts.

- Extractor: Given two accepting transcripts, as Signatures of Knowledge on message $\mathsf{PS}.pk_{i,prod}$, $(M_i, M_{i,prod}, ch_{i,prod}, s_{i,prod})$ and $(M_i, M_{i,prod}, ch'_{i,prod}, s'_{i,prod})$ the extractor computes $usk_i := (s_{i,prod} - s'_{i,prod})/(ch_{i,prod} - ch'_{i,prod})$, which is the discrete logarithm of M_i to base g_1: $M_i^{ch_{i,prod}} \cdot R_1 = g_1^{s_{i,prod}} \wedge M_i^{ch'_{i,prod}} \cdot R_1 = g_1^{s'_{i,prod}} \Longrightarrow M_i^{ch_{i,prod} - ch'_{i,prod}} = g_1^{s_{i,prod} - s'_{i,prod}} \Longrightarrow M_i = g_1^{(s_{i,prod} - s'_{i,prod})/(ch_{i,prod} - ch'_{i,prod})}$. Analogously on can argue for the discrete logarithm of $M_{i,prod}$ to base $\mathcal{H}_1(i, prod)$. □

Lemma 3.3. *The value σ output in Protocol $\Pi_{\mathsf{RS}}.\mathsf{Rate}$ is a Signature of Knowledge on message m proving the following statement:*

$$\begin{aligned}
ZKPK\{(usk_i, \sigma_i, \sigma_{i,j,prod})\colon\ & \mathsf{Verify}(\mathsf{PS}.pk, usk_i, (T_1, T_2)) = 1 \\
& \wedge \mathsf{Verify}(\mathsf{PS}.pk_{j,prod}, usk_i, (T_3, T_4)) = 1 \\
& \wedge T_5 = \mathcal{H}_1(j, prod)^{usk_i}\}.
\end{aligned}$$

Proof.

- Completeness: An honest prover will generate an accepting transcript as the verification equations hold:

$$\begin{align}
\mathrm{e}(T_1, \tilde{X})^{ch} \cdot \mathrm{e}(T_2, \tilde{g})^{-ch} \cdot \mathrm{e}(T_1, \tilde{Y})^{s} &= \mathrm{e}(T_1, \tilde{g}^{\xi_1})^{ch} \cdot \mathrm{e}(T_2, \tilde{g})^{-ch} \cdot \mathrm{e}(T_1, \tilde{g}^{\xi_2})^{k+ch \cdot usk_i} \notag \\
&= \mathrm{e}(T_1, \tilde{g}^{\xi_1})^{ch} \cdot \mathrm{e}(T_1, \tilde{g}^{\xi_2 \cdot usk_i})^{ch} \cdot \mathrm{e}(T_2, \tilde{g})^{-ch} \cdot \mathrm{e}(T_1, \tilde{g}^{\xi_2})^{k} \notag \\
&= \mathrm{e}(T_1, \tilde{g}^{\xi_1 + \xi_2 \cdot usk_i})^{ch} \cdot \mathrm{e}(T_2, \tilde{g})^{-ch} \cdot \mathrm{e}(T_1, \tilde{g}^{\xi_2})^{k} \notag \\
&= \mathrm{e}(T_1, \tilde{Y})^{k} = R_1 \tag{4} \\
\iff \mathsf{PS}.\mathsf{Verify}(&\mathsf{PS}.pk, usk_i, (T_1, T_2)) = 1 \tag{5}
\end{align}$$

$$\begin{align}
&\mathrm{e}(T_3, \tilde{X}_{j,prod})^{ch} \cdot \mathrm{e}(T_4, \tilde{g}_{j,prod})^{-ch} \cdot \mathrm{e}(T_3, \tilde{Y}_{j,prod})^{s} \notag \\
&\quad = \mathrm{e}(T_3, \tilde{g}_{j,prod}^{\xi_{1_{j,prod}}})^{ch} \cdot \mathrm{e}(T_4, \tilde{g}_{j,prod})^{-ch} \cdot \mathrm{e}(T_3, \tilde{g}_{j,prod}^{\xi_{2_{j,prod}}})^{k+ch \cdot usk_i} \notag \\
&\quad = \mathrm{e}(T_3, \tilde{g}_{j,prod}^{\xi_{1_{j,prod}}})^{ch} \cdot \mathrm{e}(T_3, \tilde{g}_{j,prod}^{\xi_{2_{j,prod}} \cdot usk_i})^{ch} \cdot \mathrm{e}(T_4, \tilde{g}_{j,prod})^{-ch} \cdot \mathrm{e}(T_3, \tilde{g}_{j,prod}^{\xi_{2_{j,prod}}})^{k} \notag \\
&\quad = \mathrm{e}(T_3, \tilde{g}_{j,prod}^{\xi_{1_{j,prod}} + \xi_{2_{j,prod}} \cdot usk_i})^{ch} \cdot \mathrm{e}(T_4, \tilde{g}_{j,prod})^{-ch} \cdot \mathrm{e}(T_3, \tilde{g}_{j,prod}^{\xi_{2_{j,prod}}})^{k} \notag \\
&\quad = \mathrm{e}(T_3, \tilde{Y}_{j,prod})^{k} = R_2 \tag{6} \\
&\iff \mathsf{PS}.\mathsf{Verify}(\mathsf{PS}.pk_{j,prod}, usk_i, (T_3, T_4)) = 1 \tag{7}
\end{align}$$

$$T_5^{-ch} \cdot \mathcal{H}_1(j, prod)^{s} = \mathcal{H}_1(j, prod)^{-usk_i \cdot ch} \cdot \mathcal{H}_1(j, prod)^{k + usk_i \cdot ch} = \mathcal{H}_1(j, prod)^{k} = R_3 \tag{8}$$

  and hence

$$ch = \mathcal{F}_{\mathsf{RO}}(T_1, T_2, T_3, T_4, T_5, R_1, R_2, R_3, prod, ppk, m). \tag{9}$$

- Simulator: Given $pp, j, prod, ppk$ and m as input and using the random oracle $\mathcal{F}_{\mathsf{RO}}$, transcripts can be simulated, as follows:

  Choose $ch, s \underset{u}{\leftarrow} \mathbb{Z}_p$ and $T_1, T_2, T_3, T_4, T_5 \underset{u}{\leftarrow} \mathbb{G}_1$, set $R_1 \coloneqq \mathrm{e}(T_1, \tilde{X})^{ch} \cdot \mathrm{e}(T_2, \tilde{g})^{-ch} \cdot \mathrm{e}(T_1, \tilde{Y})^{s}$, $R_2 \coloneqq \mathrm{e}(T_3, \tilde{X}_{j,prod})^{ch} \cdot \mathrm{e}(T_4, \tilde{g}_{j,prod})^{-ch} \cdot \mathrm{e}(T_3, \tilde{Y}_{j,prod})^{s}$, $R_3 \coloneqq T_5^{-ch} \cdot \mathcal{H}_1(j, prod)^{s}$, and patch $\mathcal{F}_{\mathsf{RO}}(T_1, T_2, T_3, T_4, T_5, R_1, R_2, R_3, prod, ppk, m) \coloneqq ch$. Under the assumption that the Decisional Diffie-Hellman Problem is hard in $\mathbb{G}_1$ the tuples (T_1, T_2), and (T_3, T_4) are indistinguishable from real signatures on message usk_i, under the respective public keys $\mathsf{PS}.pk$ and $\mathsf{PS}.pk_{j,prod}$. Futhermore, the value T_5 is chosen uniformly at random. Hence, the tuple $(g_1, M_i, \mathcal{H}_1(j, prod), T_5)$ is indistinguishable from real transcripts (all values M_i are given by $\mathcal{F}_{\mathsf{CA}}$ and hence known to a verifier). The remainder of the transcript is simulated perfectly.

- Extractor: Given two accepting transcripts $(T_1, T_2, T_3, T_4, T_5, ch, s)$ and $(T_1, T_2, T_3, T_4, T_5, ch', s')$ the extractor computes $usk_i \coloneqq (s - s')/(ch - ch')$, which is the discrete logarithm of T_5 to base $\mathcal{H}_1(j, prod)$: $T_5^{-ch} \cdot \mathcal{H}_1(j, prod)^s = R_3 \wedge T_5^{-ch'} \cdot \mathcal{H}_1(j, prod)^{s'} = R_3 \Longrightarrow T_5^{ch-ch'} = \mathcal{H}_1(j, prod)^{s-s'} \Longrightarrow T_5 = \mathcal{H}_1(j, prod)^{(s-s')/(ch-ch')}$. The tuples (T_1, T_2) and (T_3, T_4) are valid signatures on message usk_i and do not need further extraction. □

Lemma 3.4. *The value τ output in Protocol $\Pi_{\mathsf{RS}}.\mathsf{OProof}$ is a Signature of Knowledge on message (σ, i, M_i) proving the following statement:*

$$ZKPK\{(\beta, \tilde{Y}_i)\colon ct = \mathsf{CS.Enc}(\mathsf{CS}.pk, \tilde{Y}_i; \beta) \wedge \mathrm{e}(T_5, \tilde{Y}) = \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{Y}_i).$$

Proof.

- Completeness: An honest prover will generate an accepting transcript as the verification equations hold:

$$ct_1^{-\hat{ch}} \cdot g_2^{\hat{s}} = g_2^{-\beta \cdot \hat{ch}} \cdot g_2^{r + \hat{ch} \cdot \beta} = g_2^r = R_1 \tag{10}$$

$$ct_2^{-\hat{ch}} \cdot \tilde{h}^{\hat{s}} = \tilde{h}^{-\beta \cdot \hat{ch}} \cdot \tilde{h}^{r + \hat{ch} \cdot \beta} = \tilde{h}^r = R_2 \tag{11}$$

$$\begin{aligned}
&\mathrm{e}(\mathcal{H}_1(j, prod), ct_3)^{-\hat{ch}} \cdot \mathrm{e}(T_5, \tilde{Y})^{\hat{ch}} \cdot \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{f})^{\hat{s}} \\
&= \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{Y}_i \cdot \tilde{f}^{\beta})^{-\hat{ch}} \cdot \mathrm{e}(\mathcal{H}_1(j, prod)^{usk_i}, \tilde{Y})^{\hat{ch}} \cdot \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{f})^{r + \hat{ch} \cdot \beta} \\
&= \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{Y}_i^{-1} \cdot \tilde{f}^{-\beta})^{\hat{ch}} \cdot \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{Y}^{usk_i})^{\hat{ch}} \cdot \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{f})^{r + \hat{ch} \cdot \beta} \\
&= \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{Y}_i^{-1} \cdot \tilde{f}^{-\beta} \cdot \tilde{Y}^{usk_i})^{\hat{ch}} \cdot \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{f}^{\beta})^{\hat{ch}} \cdot \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{f})^{r} \\
&= \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{Y}_i^{-1} \cdot \tilde{f}^{-\beta} \cdot \tilde{Y}^{usk_i} \cdot \tilde{f}^{\beta})^{\hat{ch}} \cdot \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{f})^{r} \\
&= \mathrm{e}(\mathcal{H}_1(j, prod), 1_{\mathbb{G}_2})^{\hat{ch}} \cdot \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{f})^{r} = \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{f})^{r} = R_3
\end{aligned} \tag{12}$$

$$\begin{aligned}
&\omega \coloneqq \mathcal{H}(ct_1, ct_2, ct_3) \\
&\quad ct_4^{-\hat{ch}} \cdot (\tilde{b} \cdot \tilde{d}^{\omega})^{\hat{s}} = (\tilde{b} \cdot \tilde{d}^{\omega})^{-\beta \cdot \hat{ch}} \cdot (\tilde{b} \cdot \tilde{d}^{\omega})^{r + \hat{ch} \cdot \beta} = (\tilde{b} \cdot \tilde{d}^{\omega})^{r} = R_4
\end{aligned} \tag{13}$$

$$\begin{aligned}
\mathrm{e}(g_1, ct_3)^{-\hat{ch}} \cdot \mathrm{e}(M, \tilde{Y})^{\hat{ch}} \cdot \mathrm{e}(g_1, \tilde{f})^{\hat{s}} &= \mathrm{e}(g_1, \tilde{Y}_i \cdot \tilde{f}^{\beta})^{-\hat{ch}} \cdot \mathrm{e}(g_1^{usk_i}, \tilde{Y})^{\hat{ch}} \cdot \mathrm{e}(g_1, \tilde{f})^{r + \hat{ch} \cdot \beta} \\
&= \mathrm{e}(g_1, \tilde{Y}_i^{-1} \cdot \tilde{f}^{-\beta})^{\hat{ch}} \cdot \mathrm{e}(g_1, \tilde{Y}^{usk_i})^{\hat{ch}} \cdot \mathrm{e}(g_1, \tilde{f})^{r + \hat{ch} \cdot \beta} \\
&= \mathrm{e}(g_1, \tilde{Y}_i^{-1} \cdot \tilde{f}^{-\beta} \cdot \tilde{Y}^{usk_i})^{\hat{ch}} \cdot \mathrm{e}(g_1, \tilde{f})^{r + \hat{ch} \cdot \beta} \\
&= \mathrm{e}(g_1, \tilde{Y}_i^{-1} \cdot \tilde{f}^{-\beta} \cdot \tilde{Y}^{usk_i})^{\hat{ch}} \cdot \mathrm{e}(g_1, \tilde{f}^{\beta})^{\hat{ch}} \cdot \mathrm{e}(g_1, \tilde{f})^{r} \\
&= \mathrm{e}(g_1, \tilde{Y}_i^{-1} \cdot \tilde{f}^{-\beta} \cdot \tilde{Y}^{usk_i} \cdot \tilde{f}^{\beta})^{\hat{ch}} \cdot \mathrm{e}(g_1, \tilde{f})^{r} \\
&= \mathrm{e}(g_1, 1_{\mathbb{G}_2})^{\hat{ch}} \cdot \mathrm{e}(g_1, \tilde{f})^{r} = \mathrm{e}(g_1, \tilde{f})^{r} = R_5,
\end{aligned} \tag{14}$$

where Equation 12 holds, if and only if $\mathrm{dlog}_{\tilde{Y}}(\tilde{Y}_i) = \mathrm{dlog}_{\mathcal{H}_1(j, prod)}(T_5)$ and Equation 14 holds, if and only if $\mathrm{dlog}_{g_1}(M) = \mathrm{dlog}_{\tilde{Y}}(\tilde{Y}_i)$.

- Simulator: Given $j, \mathit{prod}, \mathit{ppk}$, and a rating (m, σ) as input and using the random oracle $\mathcal{F}_{\mathsf{RO}}$, transcripts can be simulated, as follows:

  Choose $\hat{ch}, \hat{s} \xleftarrow{u} \mathbb{Z}_p$ and $ct_1, ct_2, ct_3, ct_4 \xleftarrow{u} \mathbb{G}_2$, compute $R_1 := ct_1^{-\hat{ch}} \cdot g_2^{\hat{s}}$, $R_2 := ct_2^{-\hat{ch}} \cdot \tilde{h}^{\hat{s}}$, $R_3 := \mathrm{e}(\mathcal{H}_1(j, \mathit{prod}), ct_3)^{-\hat{ch}} \cdot \mathrm{e}(T_5, \tilde{Y})^{\hat{ch}} \cdot \mathrm{e}(\mathcal{H}_1(j, \mathit{prod}), \tilde{f})^{\hat{s}}$, $\omega := \mathcal{H}(ct_1, ct_2, ct_3)$, $R_4 := ct_4^{-\hat{ch}} \cdot (\tilde{b} \cdot \tilde{d}^{\omega})^{\hat{s}}$, $R_5 := \mathrm{e}(g_1, c_3)^{-\hat{ch}} \cdot \mathrm{e}(M, \tilde{Y})^{\hat{ch}} \cdot \mathrm{e}(g_1, \tilde{f})^{\hat{s}}$,

  and patch $\mathcal{F}_{\mathsf{RO}}(ct_1, ct_2, ct_3, ct_4, R_1, R_2, R_3, R_4, R_5, \sigma, i, M) := \hat{ch}$. The ciphertext $ct = (ct_1, ct_2, ct_3, ct_4)$ is indistinguishable from real ciphertexts, assuming the Decisional Diffie-Hellman Problem is hard in $\mathbb{G}_1$ and the remainder of the transcript is simulated perfectly.

- Extractor: Given two accepting transcripts $(ct_1, ct_2, ct_3, ct_4, \hat{ch}, \hat{s})$ and $(ct_1, ct_2, ct_3, ct_4, \hat{ch}', \hat{s}')$ the extractor computes $\beta := (\hat{s} - \hat{s}')/(\hat{ch} - \hat{ch}')$, which is the discrete logarithm of ct_1 to base g_2: $ct_1^{-\hat{ch}} \cdot g_2^{\hat{s}} = R_1 \wedge ct_1^{-\hat{ch}'} \cdot g_2^{\hat{s}'} = R_1 \Longrightarrow g_2^{\hat{s}-\hat{s}'} = ct_1^{\hat{ch}-\hat{ch}'} \Longrightarrow g_2^{(\hat{s}-\hat{s}')/(\hat{ch}-\hat{ch}')} = ct_1$. Analogously, one can argue for R_2 and R_4. Further, the extractor computes $\tilde{Y}_i := ct_3^{-\beta}$, which is the encrypted value: dividing two instances of Equation 14 gives

$$\begin{aligned} & \mathrm{e}(M, \tilde{Y})^{\hat{ch}-\hat{ch}'} \cdot \mathrm{e}(g_1, \tilde{f})^{\hat{s}-\hat{s}'} = \mathrm{e}(g_1, ct_3)^{\hat{ch}-\hat{ch}'} \\ \Longleftrightarrow\ & \mathrm{e}(M, \tilde{Y}) \cdot \mathrm{e}(g_1, \tilde{f})^{(\hat{s}-\hat{s}')/(\hat{ch}-\hat{ch}')} = \mathrm{e}(g_1, ct_3) \\ \Longleftrightarrow\ & \mathrm{e}(M, \tilde{Y}) \cdot \mathrm{e}(g_1, \tilde{f})^{\beta} = \mathrm{e}(g_1, ct_3) \\ \Longleftrightarrow\ & \mathrm{e}(M, \tilde{Y}) = \mathrm{e}(g_1, ct_3) \cdot \mathrm{e}(g_1, \tilde{f})^{-\beta} = \mathrm{e}(g_1, ct_3 \cdot \tilde{f}^{-\beta}), \end{aligned}$$

  which means that ct_3 encrypts an element $\tilde{Y}' \in \mathbb{G}_2$ such that $\mathrm{e}(M, \tilde{Y}) = \mathrm{e}(g_1, \tilde{Y}')$. The only element $\tilde{Y}'$ with this property is $\tilde{Y}_i = \tilde{Y}^{usk_i}$. □

3.4.2 The Simulator $\mathcal{S}$

$\mathcal{S}$ manages the same lists as $\mathcal{F}_{\mathsf{RS}}$, namely $\mathfrak{Params}$, $\mathfrak{Reg}$, $\mathfrak{Prods}$, $\mathfrak{Purch}$, $\mathfrak{Ratings}$, and $\mathfrak{Open}$ which are initially empty. Lists indexed with an "s" additionally store secret key material.

Simulation of $\mathcal{F}_{\mathsf{RO}}$: $\mathcal{S}$ manages the list L_{RO} and answers to requests exactly the same way as $\mathcal{F}_{\mathsf{RO}}$. At some points during the simulation - while generating anonymous ratings and opening-proofs - $\mathcal{S}$ will have to patch the list L_{RO} because $\mathcal{S}$ simulates Σ-protocols that were transformed into non-interactive zero-knowledge proofs using the Fiat-Shamir heuristic. How $\mathcal{S}$ handles this cases is described later in detail.

Simulation of $\mathcal{F}_{\mathsf{CRS}}$: $\mathcal{S}$ chooses $(\mathbb{GD}, \mathsf{PD}.pk, \mathcal{H}, \mathcal{H}_1, \mathcal{H}_2)$ according to the definition of $\mathcal{F}_{\mathsf{CRS}}$ and hands $CRS := (\mathbb{GD}, \mathsf{PD}.pk, \mathcal{H}, \mathcal{H}_1, \mathcal{H}_2)$ to every party requesting it.

Simulation of $\mathcal{F}_{\mathsf{CA}}$: $\mathcal{S}$ manages the lists L_{CA} and L_{sCA}. Whenever an honest party P_i is activated for the first time, $\mathcal{S}$ chooses $usk_i \xleftarrow{u} \mathbb{Z}_p$, computes $M_i := g_1^{usk_i}$ and sets $L_{CA}.\mathsf{Add}(P_i, M_i)$ and $L_{sCA}\mathsf{Add}(P_i, M_i, usk_i)$. When $\mathcal{S}$ receives $(\mathsf{Register}, P_i, v)$ from some (corrupted) party P_i for the first time, the tuple (P_i, v) is added to the list L_{CA}. All later $\mathsf{Register}$-requests from the same party are ignored. When $\mathcal{S}$ receives $(\mathsf{Retrieve}, P_i)$ from some party P and a tuple (P_i, v) is stored in L_{CA}, $\mathcal{S}$ sends $(\mathsf{Retrieve}, P_i, v)$ to P. If no such tuple could be found in L_{CA}, $\mathcal{S}$ sends $(\mathsf{Retrieve}, P_i, \perp)$ to P.

Simulation of Registry Key Generation: When $\mathcal{S}$ receives $(\mathsf{KeyGen}, \mathit{sid})$ from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ executes the protocol in behalf of P_{IDM}, sets $\mathfrak{Params}.\mathsf{Add}(\mathit{pp})$, $\mathfrak{Params}_s.\mathsf{Add}(\mathit{pp}, \mathit{idmsk})$ and sends $(\mathsf{KeyGen}, \mathit{sid}, \mathit{pp})$ to $\mathcal{F}_{\mathsf{RS}}$.

Simulation of User Registration:

- P_{IDM} and P_i honest: When $\mathcal{S}$ receives $(\mathsf{Register}, \mathit{sid}, \mathit{pp}', P_i)$ from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ executes the registration protocol in behalf of P_i and P_{IDM}. $\mathcal{S}$ can do this by using $(P_i, M_i, \mathit{usk}_i) \in L_{sCA}$ and $(\mathit{pp}', \mathit{idmsk}') \in \mathfrak{Params}_s$.
- P_{IDM} honest and P_i corrupted: When $\mathcal{S}$ receives (pp', R) from $\mathcal{A}$ as it intends to send from P_i to P_{IDM}, $\mathcal{S}$ sends $(\mathsf{Register}, \mathit{sid}, \mathit{pp}')$ in behalf of P_i to $\mathcal{F}_{\mathsf{RS}}$, receives $(\mathsf{Register}, \mathit{sid}, \mathit{pp}', P_i)$ from $\mathcal{F}_{\mathsf{RS}}$, and executes the protocol in behalf of P_{IDM} using $(\mathit{pp}', \mathit{idmsk}') \in \mathfrak{Params}_s$. When $\mathcal{F}_{\mathsf{RS}}$ outputs $(\mathsf{Register}, \mathit{sid}, \mathit{pp}', P_i, f)$ to P_i, $\mathcal{S}$ does not deliver this message because a corrupted P_i does not expect to receive this message from $\mathcal{F}_{\mathsf{RS}}$.
- P_{IDM} corrupted and P_i honest: When $\mathcal{S}$ receives $(\mathsf{Register}, \mathit{sid}, \mathit{pp}', P_i)$ from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ executes the protocol in behalf of P_i by using $(P_i, M_i, \mathit{usk}_i) \in L_{sCA}$. If $\mathcal{S}$ receives a value σ_i from P_{IDM} and $\mathsf{PS}.\mathsf{Verify}(\mathit{pp}', \mathit{usk}_i, \sigma_i) = 1$, set $\mathfrak{Reg}.\mathsf{Add}(P_i, \mathit{pp}', M_i, \tilde{Y}_i, \sigma_i)$ and output $(\mathsf{Register}, \mathit{sid}, \mathit{pp}', P_i, 1)$ to $\mathcal{F}_{\mathsf{RS}}$. When $\mathcal{F}_{\mathsf{RS}}$ outputs $(\mathsf{Register}, \mathit{sid}, \mathit{pp}', P_i, f)$ to P_{IDM}, $\mathcal{S}$ does not deliver this message because a corrupted P_{IDM} does not expect to receive this message from $\mathcal{F}_{\mathsf{RS}}$.

Simulation of Product Addition: When $\mathcal{S}$ receives $(\mathsf{NewProduct}, \mathit{sid}, P_i, \mathit{prod})$ from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ executes the protocol in behalf of P_i using $(P_i, M_i, \mathit{usk}_i) \in L_{sCA}$. The request to $\mathcal{F}_{\mathsf{RO}}$ does not require any special handling. After completing the protocol, $\mathcal{S}$ sets $\mathfrak{Prods}.\mathsf{Add}(P_i, \mathit{prod}, \mathit{ppk}_{i,\mathit{prod}})$, $\mathfrak{Prods}_s.\mathsf{Add}(P_i, \mathit{prod}, \mathit{ppk}_{i,\mathit{prod}}, \mathsf{PS}.\mathit{sk}_{i,\mathit{prod}})$ and outputs $(\mathsf{NewProduct}, \mathit{sid}, P_i, \mathit{prod}, \mathit{ppk}_{i,\mathit{prod}})$ to $\mathcal{F}_{\mathsf{RS}}$.

Simulation of Purchasing a Product:

- P_i and P_j honest: When $\mathcal{S}$ receives $(\mathsf{Purchase}, \mathit{sid}, P_i, P_j, \mathit{prod}, \mathit{ppk})$ from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ executes the purchasing protocol in behalf of P_i and P_j. $\mathcal{S}$ can do this by using $(P_i, M_i, \mathit{usk}_i), (P_j, M_j, \mathit{usk}_j) \in L_{sCA}$ and $(P_j, \mathit{prod}, \mathit{ppk}_{j,\mathit{prod}}, \mathsf{PS}.\mathit{sk}_{j,\mathit{prod}}) \in \mathfrak{Prods}_s$.
- P_i honest and P_j corrupted: When $\mathcal{S}$ receives $(\mathsf{Purchase}, \mathit{sid}, P_i, P_j, \mathit{prod}, \mathit{ppk})$ from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ executes the purchasing protocol in behalf of P_i, including the request to $\mathsf{VfyProd}$. $\mathcal{S}$ can do this by using $(P_i, M_i, \mathit{usk}_i) \in L_{sCA}$. When $\mathcal{F}_{\mathsf{RS}}$ outputs $(\mathsf{Purchase}, \mathit{sid}, P_i, P_j, \mathit{prod}, \mathit{ppk}, f)$ to P_j, $\mathcal{S}$ does not deliver this message because a corrupted P_j does not expect to receive this message from $\mathcal{F}_{\mathsf{RS}}$.
- P_i corrupted and P_j honest: When $\mathcal{S}$ receives $(\mathit{prod}, \mathit{ppk}, R)$ from $\mathcal{A}$ as it intends to send from P_i to P_j, $\mathcal{S}$ sends $(\mathsf{Purchase}, \mathit{sid}, P_j, \mathit{prod}, \mathit{ppk})$ in behalf of P_i to $\mathcal{F}_{\mathsf{RS}}$, receives $(\mathsf{Purchase}, \mathit{sid}, P_i, P_j, \mathit{prod}, \mathit{ppk})$ from $\mathcal{F}_{\mathsf{RS}}$, and executes the purchasing protocol in

behalf of P_j. $\mathcal{S}$ can do this by using $(P_j,\ M_j,\ usk_j) \in L_{sCA}$ and $(P_j,\ prod,\ ppk_{j,prod},$ $\mathsf{PS}.sk_{j,prod}) \in \mathfrak{Prods}_s$. When $\mathcal{F}_{\mathsf{RS}}$ outputs $(\mathsf{Purchase}, sid, P_i, P_j, prod, ppk, f)$ to P_i, $\mathcal{S}$ does not deliver this message because a corrupted P_i does not expect to receive this message from $\mathcal{F}_{\mathsf{RS}}$.

Simulation of $\mathsf{VfyProd}$: When $\mathcal{S}$ receives $(\mathsf{VfyProd}, sid, P_j, prod, ppk)$ from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ executes $\mathsf{VfyProd}(P_j, prod, ppk)$ as a local algorithm and responds as defined in the protocol. If $(P_j, M_j) \notin L_{CA}$, $\mathcal{S}$ does not respond to this request, which means that $\mathcal{F}_{\mathsf{RS}}$ waits infinitely for a response.

Remark: This product-verification is done whenever a rating is verified. It implies that all subsequent requests based on the product-verification are ignored, whenever this verification cannot be executed due to missing parameters $(P_j, M_j) \notin L_{CA}$.

Simulation of Rating a Product:

- P_{IDM} is honest: In this case $\mathcal{S}$ cannot simply execute the protocol because the identity of the rater is not known. Hence, $\mathcal{S}$ computes an accepting transcript of the underlying Σ-protocol, as follows: When receiving $(\mathsf{Rate}, sid, pp, P_j, prod, ppk, m)$ from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ uses the Zero-Knowledge simulator given in the proof of Lemma 3.3. During the simulation, $\mathcal{S}$ tries to patch the random oracle. If there is some value v such that $\big[(T_1, T_2, T_3, T_4, T_5, R_1, R_2, R_3, prod, ppk, m), v\big] \in L_{RO}$ then $\mathcal{S}$ outputs `error` and halts. Otherwise, $\mathcal{S}$ sets $L_{RO}.\mathsf{Add}\big[(T_1, T_2, T_3, T_4, T_5, R_1, R_2, R_3, prod, ppk, m), ch\big]$, $\sigma := (T_1,\ T_2,\ T_3,\ T_4,\ T_5,\ ch,\ s)$, $\mathfrak{Ratings}.\mathsf{Add}(pp, \perp, P_j, prod, ppk, m, \sigma)$ and outputs $(\mathsf{Rate}, sid, pp, P_j, prod, ppk, m, \sigma)$ to $\mathcal{F}_{\mathsf{RS}}$.
- P_{IDM} is corrupted: When receiving $(\mathsf{Rate}, sid, pp, P_i, P_j, prod, ppk, m)$ from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ generates an accepting transcript of the underlying Σ-protocol as a rating by using $(P_i, M_i, usk_i) \in L_{sCA}$, $(P_i, pp, M_i, \tilde{Y}_i, \sigma_i) \in \mathfrak{Reg}$, and $(P_i, P_j, prod, ppk, \sigma_{i,j,prod})$ - as defined in the rating protocol. Finally, $\mathcal{S}$ sets $\mathfrak{Ratings}.\mathsf{Add}(pp, P_i, P_j, prod, ppk, m, \sigma)$ and outputs $(\mathsf{Rate},\ sid,\ pp,\ P_i,\ P_j,\ prod,\ ppk,\ m,\ \sigma)$ to $\mathcal{F}_{\mathsf{RS}}$.

Simulation of Rating Verification: When $\mathcal{S}$ receives $(\mathsf{Verify},\ sid,\ pp,\ P_j,\ prod,\ ppk,\ m,\ \sigma)$ from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ executes the verification protocol. Once the value f is obtained from the protocol (which implies that $\mathsf{VfyProd}$ returned 1), $\mathcal{S}$ tries to determine the author of the rating:

Parse σ as $(T_1, T_2, T_3, T_4, T_5, ch, s)$.

If $f = 0$ **then** $\mathcal{S}$ outputs $(\mathsf{Verify}, sid, pp, P_j, prod, ppk, m, \sigma, 0, \perp)$ to $\mathcal{F}_{\mathsf{RS}}$. As defined in $\mathcal{F}_{\mathsf{RS}}$, invalid ratings will never be opened.

Else If $(pp, \perp, P_j, prod, ppk, m, \sigma) \in \mathfrak{Ratings}$ **then** output $(\mathsf{Verify},\ sid,\ pp,\ P_j,\ prod,\ ppk,\ m,\ \sigma,\ 1,\ \perp)$ to $\mathcal{F}_{\mathsf{RS}}$. In this case, P_{IDM} and the author of the rating P_i are honest - $\mathcal{F}_{\mathsf{RS}}$ will include the correct identity for the rating.

Else If P_{IDM} is honest and there exists a tuple $(P_k, pp, M_k, \tilde{Y}_k, \sigma_k) \in \mathfrak{Reg}$ such that $\mathrm{e}(T_5, \tilde{Y}) = \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{Y}_k)$ **then** Output $(\mathsf{Verify},\ sid,\ pp,\ P_j,\ prod,\ ppk,\ m,\ \sigma,\ 1,\ P_k)$. This covers the case that a rating was created by a corrupted party, while P_{IDM} is honest. Hence, $\mathcal{S}$ can use the registration information to determine the raters identity.

Else Output $(\mathsf{Verify}, \mathit{sid}, \mathit{pp}, P_j, \mathit{prod}, \mathit{ppk}, m, \sigma, 1, \perp)$ to $\mathcal{F}_{\mathsf{RS}}$. In the case that P_{IDM} is corrupted, $\mathcal{S}$ is not requested to output the correct identity.

Simulation of Linking Ratings: When $\mathcal{S}$ receives $(\mathsf{Link}$, sid, pp, P_j, prod, ppk, m_1, σ_1, m_2, $\sigma_2)$ from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ executes the linking protocol as defined in Protocol $\Pi_{\mathsf{RS}}.\mathsf{Link}$ and outputs $(\mathsf{Link}, \mathit{sid}, \mathit{pp}, P_j, \mathit{prod}, \mathit{ppk}, m_1, \sigma_1, m_2, \sigma_2, b)$ to $\mathcal{F}_{\mathsf{RS}}$, where b is the bit computed during the protocol execution. This implies that $\mathsf{VfyProd}$ returned 1.

Simulation of Determining the Raters Identity: Since $\mathcal{F}_{\mathsf{RS}}$ already asked $\mathcal{S}$ for the raters identity during verification, $\mathcal{S}$ is not involved in this step. Hence, $\mathcal{S}$ does not need to simulate something.

Simulation of Generating Opening-Proofs: When $\mathcal{S}$ receives $(\mathsf{OProof}$, sid, pp, P_j, prod, ppk, m, σ, $P)$ from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ computes $f := (\mathsf{Verify}$, sid, pp, P_j, prod, ppk, m, $\sigma)$. If $f = 0$, $\mathcal{S}$ sends $(\mathsf{OProof}$, sid, pp, P_j, prod, ppk, m, σ, P, $\perp)$ to $\mathcal{F}_{\mathsf{RS}}$. Otherwise, it is possible that $(\mathit{pp}, \perp, P_j, \mathit{prod}, \mathit{ppk}, m, \sigma) \in \mathfrak{Ratings}$ - meaning $\mathcal{S}$ simulated this rating for an honest but unknown party. In this case, $\mathcal{S}$ has to simulate an opening-proof such that P is accepted as the author of the rating. To do so, $\mathcal{S}$ uses the tuple $(P, M) \in L_{CA}$, which has to exist because the rating is valid $(f = 1)$, and executes the Zero-Knowledge simulator given in the proof of Lemma 3.4. During the simulation, $\mathcal{S}$ tries to patch the random oracle. If there is some value v such that $\big[(\mathit{ct}_1$, ct_2, ct_3, ct_4, R_1, R_2, R_3, R_4, R_5, σ, i, $M)$, $v\big] \in L_{RO}$ then $\mathcal{S}$ outputs `error` and halts. Otherwise, $\mathcal{S}$ sets $L_{RO}.\mathsf{Add}\big[(\mathit{ct}_1$, ct_2, ct_3, ct_4, R_1, R_2, R_3, R_4, R_5, σ, i, $M)$, $\hat{ch}\big]$ and $\tau := (P, \mathit{ct}_1, \mathit{ct}_2, \mathit{ct}_3, \mathit{ct}_4, \hat{ch}, \hat{s})$ and outputs $(\mathsf{OProof}, \mathit{sid}, \mathit{pp}, P_j, \mathit{prod}, \mathit{ppk}, m, \sigma, P, \tau)$ to $\mathcal{F}_{\mathsf{RS}}$.

In the case $(\mathit{pp}, \perp, P_j, \mathit{prod}, \mathit{ppk}, m, \sigma) \notin \mathfrak{Ratings}$ $\mathcal{S}$ did not simulate the rating. Hence, $\mathcal{S}$ executes the opening protocol according to $\Pi_{\mathsf{RS}}.\mathsf{OProof}$ and outputs $(\mathsf{OProof}$, sid, pp, P_j, prod, ppk, m, σ, P, $\tau)$ to $\mathcal{F}_{\mathsf{RS}}$.

Note that $\mathcal{S}$ creates the proof for party P, even if the rating was not opened yet or P was not the author of the rating. In both cases, $\mathcal{F}_{\mathsf{RS}}$ will ignore the proof and output $(\mathsf{OProof}, \mathit{sid}, \mathit{pp}, P_j, \mathit{prod}, \mathit{ppk}, m, \sigma, P, \perp)$, as expected.

Simulation of Opening-Proof Verification: When $\mathcal{S}$ receives $(\mathsf{Judge}$, sid, pp, P_j, prod, ppk, m, σ, P, $\tau)$ from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ executes the opening-proof verification protocol as defined in Protocol $\Pi_{\mathsf{RS}}.\mathsf{Judge}$ and outputs $(\mathsf{Judge}$, sid, pp, P_j, prod, ppk, m, σ, P, τ, $f)$ to $\mathcal{F}_{\mathsf{RS}}$, where f is the bit computed during the protocol execution.

3.4.3 Hybrid game $\mathcal{G}$

In this game the ideal functionality always gives $\mathcal{S}_1$ the identifying information during rating requests, i.e. instead of sending $(\mathsf{Rate}$, sid, pp, P_j, prod, ppk, $m)$ when P_{IDM} is honest, the ideal functionality sends $(\mathsf{Rate}$, sid, pp, P_i, P_j, prod, ppk, $m)$ to $\mathcal{S}_1$ both when P_{IDM} is honest and corrupted. $\mathcal{S}_1$ works exactly as $\mathcal{S}$ except when simulating ratings for honest parties and when simulating opening-proofs. To simulate ratings - both when P_{IDM} is honest and corrupted - $\mathcal{S}_1$ executes the same protocol as $\mathcal{S}$ does for corrupted P_{IDM}, which is possible because $\mathcal{S}_1$ knows the identity of the honest rater and can use its key material to generate a rating. To simulate the opening-proof generation, $\mathcal{S}_1$ executes the same protocol as $\mathcal{S}$ does for the case $(pp, \bot, P_j, prod, ppk, m, \sigma) \notin \mathfrak{Ratings}$. Even for ratings that $\mathcal{S}_1$ created for honest parties correct opening-proofs can be generated, because $\mathcal{S}_1$ used the correct identifying information for these ratings.

3.4.4 Indistinguishability of IDEAL and $\mathcal{G}$

We need to show that ratings and opening-proofs generated by $\mathcal{S}$ and $\mathcal{S}_1$ are indistinguishable. These are the only differences of the algorithms.

The rating protocol $\mathcal{S}$ executes for honest P_{IDM} uses the zero-knowledge simulator of the Σ-protocol that underlies the signature of knowledge for rating products in Protocol Π_{RS}, extended by patching $\mathcal{F}_{\mathsf{RO}}$ to generate valid ratings (see Lemma 3.3 for details). Assuming DDH is hard in $\mathbb{G}_1$ the tuples $(\sigma_{i,1}, \sigma_{i,2}, T_1, T_2)$, $(\sigma_{i,j,prod,1}, \sigma_{i,j,prod,2}, T_3, T_4)$, and $(g_1, M_i, \mathcal{H}_1(j, prod), T_5)$ generated by $\mathcal{S}$ are random DDH-instances, whereas such tuples generated by $\mathcal{S}_1$ are DDH-tuples. Given elements $(T_1, T_2, T_3, T_4, T_5) \in \mathbb{G}_1^5$ the Σ-protocol can be simulated perfectly and $\mathcal{S}$ outputs ratings that are indistinguishable from ratings $\mathcal{S}_1$ outputs, assuming patching $\mathcal{F}_{\mathsf{RO}}$ does not fail.

Analogously to the simulation of ratings, the protocol for generating opening-proofs executed by $\mathcal{S}$ uses the zero-knowledge simulator for the Σ-protocol that underlies the signature of knowledge for generating an opening-proof in Protocol Π_{RS}, extended by patching $\mathcal{F}_{\mathsf{RO}}$ to generate valid proofs (see Lemma 3.4 for details). The Cramer-Shoup encryption scheme is CCA2-secure under the DDH assumption, which means that no adversary can distinguish a valid encryption $ct = (ct_1, ct_2, ct_3, ct_4)$ from completely random tuples. Given $(ct_1, ct_2, ct_3, ct_4) \in \mathbb{G}_2^4$ the Σ-protocol can be simulated perfectly and $\mathcal{S}$ outputs opening-proofs that are indistinguishable from opening-proofs $\mathcal{S}_1$ outputs, assuming patching $\mathcal{F}_{\mathsf{RO}}$ does not fail.

Patching $\mathcal{F}_{\mathsf{RO}}$ only fails with negligible probability, because the challenge-values $ch, \hat{ch}$ used for the simulation of the Σ-protocols are chosen uniformly and independently at random. Since L_{RO} only contains polynomially many entries, choosing some values $ch, \hat{ch}$ that are already stored in L_{RO} as v only happens with negligible probability. Hence, IDEAL and $\mathcal{G}$ are indistinguishable.

3.4.5 Indistinguishability of $\mathcal{G}$ and HYBRID

By definition, $\mathcal{S}_1$ executes exactly the same operations as honest parties do when running Protocol Π_{RS}. Hence, the only way for $\mathcal{Z}$ to distinguish between $\mathcal{G}$ and HYBRID is to force $\mathcal{F}_{\mathsf{RS}}$ to output values that differ significantly from the values output by Π_{RS}. This only happens when $\mathcal{F}_{\mathsf{RS}}$ outputs `error` and halts, whereas the Protocol Π_{RS} outputs some value, or $\mathcal{F}_{\mathsf{RS}}$ outputs 0, whereas Protocol Π_{RS} outputs 1 (or vice versa). We show that every environment $\mathcal{Z}$ can do this only with negligible probability, which results in the indistinguishability of $\mathcal{G}$ and HYBRID.

Registry Key Generation: $\mathcal{F}_{\mathsf{RS}}$ always outputs the values obtained from $\mathcal{S}_1$. Since $\mathcal{S}_1$ behaves exactly as Protocol Π_{RS}, the outputs of $\mathcal{G}$ and HYBRID are indistinguishable.

User Registration: $\mathcal{F}_{\mathsf{RS}}$ enforces the outcome of this protocol only in Steps 3 and 4. For both conditions Protocol Π_{RS} generates exactly the same outputs as $\mathcal{F}_{\mathsf{RS}}$. Hence, $\mathcal{G}$ and HYBRID are indistinguishable.

Product Addition: In Step 3, $\mathcal{F}_{\mathsf{RS}}$ outputs `error` and halts, when $\mathcal{S}_1$ outputs some $ppk_{i,prod} = (M_i, M_{i,prod}, ch_{i,prod}, s_{i,prod}, \tilde{g}_{i,prod}, \tilde{X}_{i,prod}, \tilde{Y}_{i,prod})$ that is already registered for some other party P_j or for another product $prod'$. If this happens, it must hold that $\mathcal{H}_2(i, prod) = \mathcal{H}_2(j, prod) \vee \mathcal{H}_2(i, prod) = \mathcal{H}_2(i, prod') \vee \mathcal{H}_2(i, prod) = \mathcal{H}_2(j, prod')$. In any case, this would be a collision in the collision-resistant hash function $\mathcal{H}_2$. Analogously we can argue for $\mathcal{H}_1$. Furthermore, $ppk_{i,prod}$ includes a proof of knowledge of the value usk_i, which is chosen uniformly and independently for honest parties. Hence, the probability that $ppk_{i,prod}$ is already registered for some other party or some other product is negligible, implying that $\mathcal{G}$ and HYBRID are indistinguishable.

Purchase: The purchasing request will be ignored, according to Step 2 in $\mathcal{F}_{\mathsf{RS}}$, when $P_i = P_j$ or when $\mathsf{VfyProd}$ returned 0. The same is done in Protocol Π_{RS}. If P_j is corrupted and did not register (P_j, M_j) to $\mathcal{F}_{\mathsf{CA}}$, $\mathcal{S}_1$ is not able to execute the $\mathsf{VfyProd}$-algorithm and ignores the request. This means that $\mathcal{F}_{\mathsf{RS}}$ gets no response from $\mathcal{S}_1$ and hence does not execute the purchasing protocol. That implies that $\mathcal{G}$ and HYBRID are indistinguishable.

Only in Step 4 the outcome of the purchasing protocol is fixed by $\mathcal{F}_{\mathsf{RS}}$. But in that case $\mathcal{S}_1$ knows all information needed to execute the protocol in behalf of P_i and P_j as defined in Protocol Π_{RS} and outputs 1. Hence, $\mathcal{G}$ and HYBRID are indistinguishable.

$\mathsf{VfyProd}$ The product verification is only used as a subprotocol within the purchasing protocol and all rating verifications - there is no direct activation from $\mathcal{Z}$ for this. Whenever $\mathsf{VfyProd}$ returns 0 the calling protocol will ignore the request, both in $\mathcal{F}_{\mathsf{RS}}$ and Protocol Π_{RS}. $\mathcal{S}_1$ exploits this behavior by ignoring the $\mathsf{VfyProd}$-request, when a corrupted party did not register at $\mathcal{F}_{\mathsf{CA}}$. This will in turn ignore the request to the calling protocol. Hence, $\mathcal{G}$ and HYBRID are indistinguishable in that case.

During the product verification $\mathcal{F}_{\mathsf{RS}}$ could differ in the output from Protocol Π_{RS} in Steps 2, 3, and 5. Step 2 ensures consistency. Since $\mathsf{VfyProd}$ is a deterministic algorithm in Protocol Π_{RS}, consistency is guaranteed. Step 3 covers the case that a maliciously generated ppk would be accepted as honestly generated (by an honest party). We can prove that this happens only with negligible probability, via a reduction to the PS1-Problem. In Step 5 $\mathcal{F}_{\mathsf{RS}}$ ensures that every ppk is only valid for exactly one party P_i and one product $prod$. Analogously to Product Addition, if ppk would also be valid for some party P_j and/or a product $prod'$, this breaks the collision-resistance of the hash functions $\mathcal{H}_1$ and $\mathcal{H}_2$. Hence, $\mathcal{G}$ and HYBRID are indistinguishable.

Rate a Product: To generate valid ratings $\mathcal{F}_{\mathsf{RS}}$ ensures in Step 2 that the party P_i is registered, purchased the product to rate - which implies that the product is valid - and did not rate the specified product yet. The same is checked in Protocol Π_{RS}. Step 7 covers consistency: the tuple (pp, P', P_j, $prod$, ppk, m, σ, 0, lid, oid), for some P', lid, oid, only exists in the list $\mathfrak{Ratings}$, if $(pp, P_j, prod, ppk, m, \sigma)$ is verified by $\mathcal{S}_1$ as invalid and was verified before the rating request occurred. This is because the only possibility to store σ as invalid is Step 6 during the verification of ratings in $\mathcal{F}_{\mathsf{RS}}$. During rating requests Protocol Π_{RS}, and hence the simulator $\mathcal{S}_1$, only generates valid ratings, i.e. the deterministic verification algorithm will output 1. Hence, the rating σ output by $\mathcal{S}_1$ cannot exist in $\mathfrak{Ratings}$ or is verified as valid. In both cases the tuple $(pp, P', P_j, prod, ppk, m, \sigma, 0, lid, oid)$ is not stored in $\mathfrak{Ratings}$. Hence, $\mathcal{G}$ and HYBRID are indistinguishable.

Remark: In Game $\mathcal{G}$ the simulator $\mathcal{S}_1$ always gets the identity of a rater and can generate ratings as defined in Protocol Π_{RS}.

Verifying a Rating: To verify ratings the VfyRtg-protocol is used. This subprotocol is also an essential tool during the Link, Open, OProof, and Judge protocols, because these protocols are only meaningful for valid ratings. During the VfyRtg-protocol the specified product is verified by the VfyProd-protocol. Hence, whenever $\mathcal{S}_1$ does not respond to a VfyProd request, also VfyRtg and its calling protocols will not proceed, which exploits the request ignoring behavior.

Analogously to Step 7 of the Rate protocol, Step 4 covers consistency. Since the verification algorithm in Protocol Π_{RS} is deterministic, two verification requests with the same input will generate the same output, as required by $\mathcal{F}_{\mathsf{RS}}$. The Steps 5–9 can only occur for ratings that were not generated by honest parties using the Rate-protocol.

Step 5 handles invalid ratings and self-ratings. The verification protocol in Protocol Π_{RS} covers the same cases: invalid ratings are recognized by the test $ch = \mathcal{F}_{\mathsf{RO}}(T_1, T_2, T_3, T_4, T_5, R_1, R_2, R_3, prod, ppk, m)$ and self-ratings are recognized by $T_5 \neq M_{j,prod}$. Obviously, when P_{IDM} and P_j are corrupted, it is possible to generate arbitrary tuples $(usk'_j, \sigma'_j, \sigma_{j,j,prod})$ such that $\mathsf{PS.Verify}(\mathsf{PS}.pk, usk'_j, \sigma'_j) = 1$ and $\mathsf{PS.Verify}(\mathsf{PS}.pk_{j,prod}, usk'_j, \sigma_{j,j,prod}) = 1$, but $\mathcal{H}_1(j, prod)^{usk'_j} \neq M_{j,prod}$. With these values P_j can rate his own product, such that the verification algorithm cannot detect it. But in this case $\mathcal{S}_1$ is not requested to output the identity of the signer ($P = \perp$). Hence, $\mathcal{F}_{\mathsf{RS}}$ either outputs 0 because the rating is invalid ($b = 0$), or continues executing the verification protocol. Since P_{IDM} and P_j are considered as corrupted in this case, the Steps 7–9 do not occur. Hence, $\mathcal{F}_{\mathsf{RS}}$ will output 1 in Step VfyRtg, as it would also happen in Protocol Π_{RS}. When P_{IDM} is honest and a corrupted P_j generates a valid rating $(T_1, T_2, T_3, T_4, T_5, ch, s)$ for his own product, this means that he proved knowledge of a value usk, such that (T_1, T_2) is a valid signature for message $usk \neq usk_j$ under the public key $\mathsf{PS}.pk$ from P_{IDM} and that $T_5 \neq M_{j,prod}$. Since P_{IDM} is honest, $\mathcal{S}_1$ has to output the identity of the rater. If $usk = usk_i$ for some honest party P_i, $\mathcal{S}_1$ will find an entry in $\mathfrak{Reg}$ that falsely identifies P_i as the rater. If P_{IDM} does not find an identifying entry in $\mathfrak{Reg}$, $\mathcal{S}_1$ returns $\perp$ to $\mathcal{F}_{\mathsf{RS}}$. Both cases are discussed in Steps 8 and 9.

Step 7 ensures that rating a product of an honest party P_j is only possible after purchasing it. Every rating includes a proof of knowledge of a valid signature for some message usk under the public key $\mathsf{PS}.pk_{j,prod}$ from party P_j. This signature is handed to the rater during the Purchase-protocol, but in this case some corrupted party proved knowledge of such a signature without executing the Purchase-protocol. That means, the signature must be a forgery, which contradicts the EUF-CMA security of the signature scheme in use. We will prove this in detail via a reduction to the PS1-Problem. Hence, under the PS1-Problem $\mathcal{G}$ and HYBRID are indistinguishable in that case.

Step 8 ensures strong-exculpability, meaning that it is not feasible to produce valid ratings in behalf of honest parties. Impersonating an honest party requires to find the parties' secret key usk. Using a reduction to the PS1-Problem we can prove that this is only possible with negligible probability.

Step 9 ensures traceability, meaning that the identity of every rater can be determined from valid ratings. Being able to create valid but untraceable ratings requires to forge a signature of the EUF-CMA secure signature scheme used by P_{IDM}. Analogously to Step 8, we can prove that such attacks are infeasible under the PS1-Problem.

Linking Ratings: Whenever $\mathcal{S}_1$ has to respond to a Link-request, we know that the specified product is valid (VfyProd returned 1) and $\mathcal{F}_{\mathsf{RS}}$ stored the ratings to link in the list $\mathfrak{Ratings}$. Both the simulator $\mathcal{S}_1$ and $\mathcal{F}_{\mathsf{RS}}$ return 0 to the Link request when at least one of the ratings is invalid. Hence, for the analysis of Link we only consider valid ratings that passed all verification-tests.

Now we analyze Link-requests.

Step 4 claims that ratings are unique. This is ensured by the Verify-request because $\mathcal{F}_{\mathsf{RS}}$ adds a rating only if it is not present in the list $\mathfrak{Ratings}$.

In Step 5 $\mathcal{F}_{\mathsf{RS}}$ enforces consistency. If two ratings are linkable, RebLDB is used to store this information and subsequent Link-requests for ratings of the same equivalence class must be linkable, too. In Protocol Π_{RS} for every party and a given product the element T_5 of a rating $\sigma = (T_1, T_2, T_3, T_4, T_5, ch, s)$ is a fixed value. Hence, all ratings for a fixed product with identical values T_5 belong to the same equivalence class and are linkable, as expected.

In Step 6 $\mathcal{F}_{\mathsf{RS}}$ has no information that could be used to link the given ratings. This can only happen when P_{IDM} is corrupted and the ratings are also generated by corrupted parties. So the value obtained from the simulator $\mathcal{S}_1$ is used as output. Hence, Protocol Π_{RS} and $\mathcal{F}_{\mathsf{RS}}$ generate the same output.

Step 7 expresses that it must be infeasible to generate ratings that can be opened to different parties but are linkable. Since every rating $\sigma = (T_1, T_2, T_3, T_4, T_5, ch, s)$ in Protocol Π_{RS} includes a zero-knowledge proof of knowledge of some value usk such that (T_1, T_2) and (T_3, T_4) are valid signatures for message usk and the discrete logarithm of T_5 to the base $\mathcal{H}_1(j, prod)$ is the same value usk, this requirement holds. Hence, in this case Protocol Π_{RS} and $\mathcal{F}_{\mathsf{RS}}$ generate the same output.

Step 8 covers the case that it must be infeasible to generate a valid rating in behalf of an honest user. This is analogue to VfyRtg-Step 8 for an corrupted P_{IDM} and we can also prove via a reduction to the PS1-Problem that this event does not occur.

Step 9 is analogue to Step 6.

The RebLDB-Step 18 is analogue to VfyRtg-Step 7 for corrupted P_{IDM}. With a reduction to the PS1-Problem we can prove that this event does not occur.

Summarizing the analysis of Link and RebLDB, $\mathcal{G}$ and HYBRID are indistinguishable.

Determine Raters Identity: $\mathcal{S}_1$ outputs the raters' identity within the Verify-protocol. Since $\mathcal{S}$ can correctly output the identity, also $\mathcal{S}_1$ can do this.

Generate Opening-Proofs: This activation is only of interest for honest P_{IDM}. $\mathcal{F}_{\mathsf{RS}}$ and Protocol Π_{RS} only generate an opening-proof for valid ratings that were opened to the given party. For invalid or unopened ratings and when the given rater identity is incorrect both protocols output $\perp$. Hence, for the analysis of OProof we only consider valid ratings that passed all verification-tests. Step 7 covers consistency: an opening-proof that was once invalid cannot be made valid. Since the Judge-protocol is deterministic in Protocol Π_{RS} and OProof only generates valid proofs, this cannot happen. Hence, $\mathcal{G}$ and HYBRID are indistinguishable.

Verifying Opening-Proofs: If the rating is invalid, no party identity or no opening-proof is given, $\mathcal{F}_{\mathsf{RS}}$ and Protocol Π_{RS} both output 0. For the analysis of Judge we only consider valid ratings that passed all verification-tests. Step 5 expresses that P_{IDM} is honest, the rating was generated by an honest party, or Step 16 occurred previously for this specific rating. The Steps 7 and 8 cover consistency, which is ensured by Protocol Π_{RS} because the verification of opening-proofs is deterministic. Every valid opening-proof for the correct identity will be verified as valid by Protocol Π_{RS}; invalid proofs will detected as those. Opening-proofs to a wrong identity can be detected by Protocol Π_{RS} because the real identity the proof was generated for is a part of τ. Step 9 covers non-frameability. Maliciously generated but valid opening-proofs cause $\mathcal{F}_{\mathsf{RS}}$ to output `error` and halt, when P_{IDM} and P are honest. Via a reduction to the CCA2-security of the Cramer-Shoup encryption scheme we can prove that such opening-proofs cannot be generated. Step 10 expresses that P_{IDM} is corrupted and the given rating was generated by a corrupted party. Otherwise, $\mathcal{F}_{\mathsf{RS}}$ would know the identity of the rater. In Step 12 consistency is guaranteed, as in Steps 7 and 8. In Step 13 an honest user is accepted as the author of the given rating. Since $\mathcal{F}_{\mathsf{RS}}$ does not know the identity of that rating ($X = \perp$) the rating must be maliciously generated in behalf of an honest party. This is impossible as we will prove via a reduction to PS1-Problem. In Step 15 the simulator accepts the opening-proof as valid for party P. Hence, the identity P is stored to ensure consistency for future verification requests. The Steps 16 and 17 store the verified opening-proof for the given rating to ensure consistency for future verification requests.

As discussed above, using reductions to the PS1-Problem and a reduction to the CCA2-security of the Cramer-Shoup encryption scheme, we can conclude that no environment $\mathcal{Z}$ can distinguish between $\mathcal{G}$ and HYBRID. The reductions complete the proof.

3.4.6 The Reductions used within the Security Proof

To complete the proof of Theorem 3.1 we have to show that no environment can use the Steps VfyProd.3, VfyRtg.7/8/9, RebLDB.18, LinkRtgs.8 and Judge.9/13 to its advantage. We prove this with several reductions using a proof of knowledge extractor which needs rewinding. In UC the environment is treated as an interactive distinguisher, i.e. rewinding an interactive machine is not possible. This is not contradicting because we use rewinding to prove the indistinguishability of hybrid games and not within the simulation. The same technique was used within other UC-based proofs [Lin11, BCPV13, Gro04].

All proofs have the same structure: assuming there exists an environment $\mathcal{Z}$ that can distinguish between $\mathcal{G}$ and HYBRID, and given either a PS1-instance or a CCA2-challenger for the Cramer-Shoup encryption scheme, we define a simulator interacting with $\mathcal{F}_{\mathsf{RS}}$ in game $\mathcal{G}$ we use to find a solution to the given problem instance. Since we assume the PS1-Problem and the SXDH-Assumption hold, no such environment can exist.

Lemma 3.5. *If the PS1-Problem holds for bilinear group generator* $\mathsf{BiGrGen}$*, then no environment can distinguish between* $\mathcal{G}$ *and* HYBRID *at Steps* $\mathsf{VfyProd}.3$*,* $\mathsf{VfyRtg}.8$*,* $\mathsf{LinkRtgs}.8$*, or* $\mathsf{Judge}.13$*.*

Proof. Assume that there exists an environment $\mathcal{Z}$ interacting with game $\mathcal{G}$ that is able to let $\mathcal{F}_{\mathsf{RS}}$ output `error` and halt at the Steps $\mathsf{VfyProd}.3$, $\mathsf{VfyRtg}.8$, $\mathsf{LinkRtgs}.8$, or $\mathsf{Judge}.13$ with non-negligible probability. We will use this environment to define a simulator $\mathcal{S}_2$ that we can use to compute a solution to the Pointcheval-Sanders-Problem with non-negligible probability. The hash function $\mathcal{H}_1$ is treated as a random oracle.

We are given $\mathbb{GD}$ as the output of $\mathsf{BiGrGen}$, $(g, Y, \tilde{g}, \tilde{X}, \tilde{Y})$ and unlimited access to oracle $\mathcal{O}$ from our challenger and have to output a tuple $(m^*, s, s^{x+m^* \cdot y})$ such that $s \neq 1_{\mathbb{G}_1}$ and m^* was not asked to $\mathcal{O}$. In the first part of the proof, we will describe how $\mathcal{S}_2$ interacts with $\mathcal{F}_{\mathsf{RS}}$ and handles the interaction with $\mathcal{Z}$ and the real-world adversary $\mathcal{A}$. In the second part we analyze $\mathcal{S}_2$.

Simulator $\mathcal{S}_2$ works as $\mathcal{S}_1$, except in the following cases:

Calls to $\mathcal{F}_{\mathsf{RO}}$: $\mathcal{S}_2$'s answers are generated the same way as $\mathcal{S}_1$ does. We will point out the situations in which $\mathcal{S}_2$ deviates from this policy.

Calls to $\mathcal{F}_{\mathsf{CRS}}$: $\mathcal{S}_2$ runs $\mathsf{PD}.\mathsf{KeyGen}(\mathbb{GD})$ to obtain $\mathsf{PD}.pk := (u, v)$ and $\mathsf{PD}.td := \mathrm{dlog}_u(v)$. The common reference string is set to $(\mathbb{GD}$, $\mathsf{PD}.pk$, $\mathcal{H}$, $\mathcal{H}_1$, $\mathcal{H}_2)$ according to the definition of $\mathcal{F}_{\mathsf{CRS}}$ in $\mathcal{G}$.

Calls to $\mathcal{H}_1$: $\mathcal{S}_2$ manages the list $L_{\mathcal{H}_1}$ to respond identically to repeated requests. When some x is queried for the first time ($\mathcal{H}_1(x)$ is called for some $x \in \{0,1\}^*$), $\mathcal{S}_2$ chooses $\alpha_x \xleftarrow{r} \mathbb{Z}_p$, computes $\hat{g}_x := g_1^{\alpha_x}$, and stores $(x, \alpha_x, \hat{g}_x)$ in $L_{\mathcal{H}_1}$. Finally, $\mathcal{S}_2$ hands $\hat{g}_x$ to the caller, as it is also done for repeated queries $\mathcal{H}_1(x)$, i.e. $(x, \alpha_x, \hat{g}_x) \in L_{\mathcal{H}_1}$.

Calls to $\mathcal{F}_{\mathsf{CA}}$: Whenever an honest party P_i is activated for the first time, $\mathcal{S}_2$ chooses $u_i \xleftarrow{r} \mathbb{Z}_p$, computes $M_i := Y^{u_i}$ and sets $L_{CA}.\mathsf{Add}(P_i, M_i)$. Note that the user-secret-key usk_i is implicitly set to be $y \cdot u_i$ for an unknown y. Calls from corrupted parties are handled as defined for $\mathcal{S}_1$.

Registry Key Generation: For an honest P_{IDM} $\mathcal{S}_2$ handles the KeyGen-requests as defined for $\mathcal{S}_1$. A corrupted P_{IDM} is managed by adversary $\mathcal{A}$.

User Registration: For honest P_{IDM} $\mathcal{S}_2$ works exactly as $\mathcal{S}_1$. $\mathcal{S}_2$ simulates the computations for an honest party P_i using the interactive simulator given in the proof of Lemma 3.1.

Product Addition: For honest party P_i, $\mathcal{S}_2$ sets $M_{i,prod} := Y^{u_i \cdot \alpha_{i,prod}}$ and computes $\tilde{g}_{i,prod} := \mathcal{H}_2(i, prod)$, where $\alpha_{i,prod}$ is set during the request $\mathcal{H}_1(i, prod)$. Then, $\mathcal{S}_2$ runs the algorithm $\mathsf{PS.KeyGen}(\mathbb{GD})$ to obtain $\mathsf{PS}.pk_{i,prod} := (\tilde{g}_{i,prod}, \tilde{X}_{i,prod}, \tilde{Y}_{i,prod})$ and $\mathsf{PS}.sk_{i,prod} := ({\xi_1}_{i,prod}, {\xi_2}_{i,prod})$ and simulates the non-interactive zero-knowledge proof of knowledge using the simulator given in the proof of Lemma 3.2. With these values, $\mathcal{S}_2$ runs the remaining steps defined in Protocol Π_{RS}.

Purchase: In behalf of an honest seller P_j, $\mathcal{S}_2$ behaves as $\mathcal{S}_1$. For an honest purchaser P_i, $\mathcal{S}_2$ uses the same simulator as during the $\mathsf{Register}$-protocol (see Lemma 3.1).

$\mathsf{VfyProd}$: $\mathcal{S}_2$ works exactly as $\mathcal{S}_1$.

Rate a Product: To simulate ratings for an honest party P_i (note that $\mathcal{S}_2$ obtains the identity from $\mathcal{F}_{\mathsf{RS}}$), $\mathcal{S}_2$ uses the values σ_i from the $\mathsf{Register}$-protocol with P_{IDM}, $\sigma_{i,j,prod}$ from the $\mathsf{Purchase}$-protocol with P_j, $\alpha_{j,prod}$ chosen by $\mathcal{H}_1$, u_i chosen by $\mathcal{F}_{\mathsf{CA}}$, and Y given by the PS1-instance: choose $t_1, t_2, k \leftarrow_R \mathbb{Z}_p$ and compute $T_1 := \sigma_{i,1}^{t_1}$, $T_2 := \sigma_{i,2}^{t_1}$, $T_3 := \sigma_{i,j,prod,1}^{t_2}$, $T_4 := \sigma_{i,j,prod,2}^{t_2}$, $T_5 := Y^{\alpha_{j,prod} \cdot u_i}$. With these values $\mathcal{S}_2$ simulates the zero-knowledge proof of knowledge as given in the proof of Lemma 3.3. Then $\mathcal{S}$ patches $\mathcal{F}_{\mathsf{RO}}$ by setting $L_{RO}.\mathsf{Add}(T_1, T_2, T_3, T_4, T_5, R_1, R_2, R_3, prod, ppk, m), ch)$, sets $\sigma := (T_1, T_2, T_3, T_4, T_5, ch, s)$ and outputs σ as the rating.

For all remaining protocols ($\mathsf{Verify}, \mathsf{Link}, \mathsf{Open}, \mathsf{OProof}, \mathsf{Judge}$) $\mathcal{S}_2$ works exactly as $\mathcal{S}_1$.

Now we show how $\mathcal{S}_2$ can be used to find a solution to the given PS1-instance.

When $\mathcal{F}_{\mathsf{RS}}$ outputs `error` in $\mathsf{VfyProd}.3$, we know that the party P_j is honest, ppk fulfills the verification equations defined in $\mathsf{VfyProd}$, and P_j did not use the $\mathsf{NewProduct}$-protocol to generate ppk. Especially, for $ppk = (M_j, M_{j,prod}, ch_{j,prod}, s_{j,prod}, \tilde{g}_{j,prod}, \tilde{X}_{j,prod}, \tilde{Y}_{j,prod})$ the non-interactive zero-knowledge proof of knowledge $(M_j, M_{j,prod}, ch_{j,prod}, s_{j,prod})$ is valid. Now we rewind the game $\mathcal{G}$ up to the point where $\mathcal{F}_{\mathsf{RO}}$ outputs $ch_{j,prod}$ for the first time. In the rewound game, $\mathcal{S}_2$ lets $\mathcal{F}_{\mathsf{RO}}$ output a new value $ch'_{j,prod} \neq ch_{j,prod}$. Eventually, $\mathcal{S}_2$ obtains some ppk' for the same pair $(j, prod)$ as in the first run of the game, where the non-interactive zero-knowledge proof of knowledge $(M_j, M_{j,prod}, ch'_{j,prod}, s'_{j,prod})$ is valid, too. Using the extractor of Lemma 3.2 we obtain $usk_i = y \cdot u_i$, where u_i is chosen by $\mathcal{F}_{\mathsf{CA}}$. Furthermore, we can compute $y := u_i^{-1} \cdot usk_i$ and use it to find a solution to the PS1-Problem.

When $\mathcal{F}_{\mathsf{RS}}$ outputs `error` in $\mathsf{VfyRtg}.8$, $\mathsf{LinkRtgs}.8$, or $\mathsf{Judge}.13$, the given rating $\sigma = (T_1, T_2, T_3, T_4, T_5, ch, s)$ is valid and must be maliciously generated in behalf of an honest user P, as discussed previously. We rewind the game $\mathcal{G}$ up to the point where $\mathcal{F}_{\mathsf{RO}}$ outputs c for the first time. In the rewound game, $\mathcal{S}_2$ lets $\mathcal{F}_{\mathsf{RO}}$ output a new value $ch' \neq ch$. Eventually, $\mathcal{S}_2$ obtains another valid rating $\sigma' = (T_1, T_2, T_3, T_4, T_5, ch', s')$ for the same P_j, $prod$, ppk and m. Using the extractor of Lemma 3.3 we obtain $usk = y \cdot u_i$, where u_i is chosen by $\mathcal{F}_{\mathsf{CA}}$. Analogously to $\mathsf{VfyProd}.3$, we can compute $y := u_i^{-1} \cdot usk$ and use this value to find a solution to the PS1-Problem.

To compute a solution to the PS1-Problem given the value y, we choose $m \overset{\$}{\leftarrow} \mathbb{Z}_p$, query the oracle $\mathcal{O}(m)$ and obtain a pair $(H_1, H_2) := (h, h^{x+m\cdot y}) \in \mathbb{G}_1$ for some unknown $x \in \mathbb{Z}_p$. Then we set $H_3 := H_2 \cdot H_1^{-m\cdot y} = h^{x+m\cdot y} \cdot h^{-m\cdot y} = h^x$, choose $r, m^* \overset{\$}{\leftarrow} \mathbb{Z}_p$ and output $(m^*, H_1^r, H_3^r \cdot H_1^{r\cdot m^* \cdot y})$.

All outputs from $\mathcal{S}_2$ are distributed identically to the outputs of $\mathcal{S}_1$, assuming patching the random oracles does not fail. As argued previously, this only happens with negligible probability. Hence, when $\mathcal{Z}$ can distinguish between the game $\mathcal{G}$ and HYBRID at the Steps VfyProd.3, VfyRtg.8, LinkRtgs.8, or Judge.13 we can solve the PS1-Problem with non-negligible probability. □

Lemma 3.6. *If the SXDH-Assumption holds for bilinear group generator* BiGrGen*, and hence the Cramer-Shoup encryption scheme is CCA2-secure in $\mathbb{G}_2$, then no environment can distinguish between $\mathcal{G}$ and* HYBRID *at Step* Judge*.9.*

Proof. Assume that there exists an environment $\mathcal{Z}$ interacting with game $\mathcal{G}$ that is able to let $\mathcal{F}_{\mathsf{RS}}$ output `error` at the Step Judge.9 with non-negligible probability. We will use this environment to define a simulator $\mathcal{S}_3$ which we use to break the CCA2-security of the Cramer-Shoup encryption scheme. We use the LR-formulation for CCA2-security [BR07], which is equivalent to the standard CCA2-notion.

We are given $\mathbb{GD}$ as the output of BiGrGen, $\mathsf{CS}.pk = (g_2, \tilde{h}, \tilde{b}, \tilde{d}, \tilde{f}, \mathcal{H})$, access to an encryption-oracle $\mathcal{LR}$ and access to a decryption-oracle $\mathcal{D}$ from our challenger. We have to output a bit b as a guess whether the left ($b = 0$) or the right ($b = 1$) message given to oracle $\mathcal{LR}$ was encrypted, under the limitation not to query $\mathcal{D}$ to decrypt some ciphertext produced by $\mathcal{LR}$.

In the first part of the proof, we will describe how $\mathcal{S}_3$ interacts with $\mathcal{F}_{\mathsf{RS}}$ and handles the interaction with $\mathcal{Z}$ and the real-world adversary $\mathcal{A}$. In the second part we analyze $\mathcal{S}_3$. Note that Step Judge.9 is only of interest for honest P_{IDM}. A corrupted P_{IDM} is always able to generate opening-proofs, because it controls the encryption scheme. Hence, we assume $\mathcal{A}$ does not corrupt P_{IDM}.

Simulator $\mathcal{S}_3$ works as $\mathcal{S}_1$, except in the following cases:

Registry Key Generation: $\mathcal{S}_3$ generates $\mathsf{PS}.pk$ and $\mathsf{PS}.sk$ as $\mathcal{S}_1$ does and sets the public key of the Cramer-Shoup encryption to be $\mathsf{CS}.pk$ given from the challenger.

User Registration: When the honest P_{IDM} interacts with an corrupted party P_i, $\mathcal{S}_3$ works exactly as $\mathcal{S}_1$, except when decrypting the value $ct = (ct_1, ct_2, ct_3, ct_4)$ obtained from party P_i. Here, $\mathcal{S}_3$ queries its decryption oracle $\mathcal{D}(ct)$ to obtain the value $\tilde{Y}_i$. When P_{IDM} interacts with an honest party P_i, $\mathcal{S}_3$ queries the $\mathcal{LR}$-oracle with $1_{\mathbb{G}_2}$ and $\tilde{Y}_i = \tilde{Y}^{usk_i}$ as $\mathcal{LR}(1_{\mathbb{G}_2}, \tilde{Y}_i)$ to obtain the ciphertext $ct = (ct_1, ct_2, ct_3, ct_4)$ and uses it during the protocol execution.

For the protocols NewProduct, Purchase, VfyProd, Rate, Verify, Link, and Open $\mathcal{S}_3$ works exactly as $\mathcal{S}_1$, because there is no encryption involved.

Generating Opening-Proofs: When P_{IDM} has to generate an opening-proof for a corrupted party P_i, it executes exactly the same protocol as $\mathcal{S}_1$. When P_{IDM} has to generate an opening-proof for an honest party P_i, $\mathcal{S}_3$ runs the same verification checks as $\mathcal{S}_1$ does, queries $\mathcal{LR}(1_{\mathbb{G}_2}, \tilde{Y}_i)$ to obtain the ciphertext $ct = (ct_1, ct_2, ct_3, ct_4)$ and simulates the opening-proof using the simulator of Lemma 3.4. Then $\mathcal{S}_3$ patches $\mathcal{F}_{\mathsf{RO}}$, sets $\tau := (P_i, ct_1, ct_2, ct_3, ct_4, \hat{ch}, \hat{s})$ and outputs τ.

Opening-Proof Verification: $\mathcal{S}_3$ works exactly as $\mathcal{S}_1$.

Now we show how $\mathcal{S}_3$ can be used to break the CCA2-security of the Cramer-Shoup encryption scheme.

We are working with Type-3 pairings where no map from $\mathbb{G}_1$ to $\mathbb{G}_2$ exists. Therefore, elements from $\mathbb{G}_1$ cannot be used to compute elements in $\mathbb{G}_2$ and we can concentrate on the group $\mathbb{G}_2$ during the analysis.

When $\mathcal{F}_{\mathsf{RS}}$ outputs `error` in Step 9 of $(\mathsf{Judge}, sid, pp, P_j, prod, ppk, m, \sigma, P_i, \tau)$ we know from the validity of the non-interactive zero-knowledge proofs of knowledge σ and τ that

$$T_5 = \mathcal{H}_1(j, prod)^{usk_i} \qquad (T_5 \text{ is given by } \sigma,)$$

$$ct_3 = \tilde{Z} \cdot \tilde{f}^{\beta} \qquad (ct_3 \text{ is given by } \tau,\ \beta \text{ is unknown})$$

and

$$\mathrm{e}(\mathcal{H}_1(j, prod)^{usk_i}, \tilde{Y}) = \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{Z}), \qquad (\text{since } \tau \text{ is valid})$$

which is only possible, when $\tilde{Z} = \tilde{Y}_i$. This in turn means that the opening-proof contains the correct value $\tilde{Y}_i$ for party P_i.

The ciphertexts of $\tilde{Y}_i$ that were generated during the $\mathsf{Register}$-protocol and for other opening-proofs for the same party are the only values that depend on $\tilde{Y}_i$. But these ciphertexts contain $\tilde{Y}_i$ only if the $\mathcal{LR}$-oracle encrypts the message on the right-hand side of a call $(b = 1)$. Hence, we output $b' = 1$ as our guess to the CCA2-challenger.

When $\mathcal{F}_{\mathsf{RS}}$ never outputs `error`, meaning that it was not possible to maliciously produce an opening-proof, we output $b' = 0$ as our guess to the CCA2-challenger, because we assume that $1_{\mathbb{G}_2}$ was encrypted using $\mathcal{LR}$. In this case all ciphertexts are independent of $\tilde{Y}_i$, which implies that computing $\tilde{Y}_i$ is not possible.

All outputs from $\mathcal{S}_3$ are distributed identically to the outputs of $\mathcal{S}_1$, assuming patching the random oracle does not fail. As argued previously, this only happens with negligible probability. Hence, when $\mathcal{Z}$ can distinguish between the game $\mathcal{G}$ and HYBRID at the Step $\mathsf{Judge}.9$ we can break the CCA2-security of the Cramer-Shoup encryption scheme with non-negligible probability. □

Lemma 3.7. *If the PS1-Problem holds for bilinear group generator* $\mathsf{BiGrGen}$*, then no environment can distinguish between* $\mathcal{G}$ *and* HYBRID *at Steps* VfyRtg.*7/9 and* RebLDB.*18.*

Proof. Assume that there exists an environment $\mathcal{Z}$ interacting with game $\mathcal{G}$ that is able to let $\mathcal{F}_{\mathsf{RS}}$ output `error` at the Steps $\mathsf{VfyRtg}.7$, $\mathsf{VfyRtg}.9$ or $\mathsf{RebLDB}.18$ with non-negligible probability. We will use this environment to define a simulator $\mathcal{S}_4$ that we can use to compute a solution to the Pointcheval-Sanders-Problem with non-negligible probability. The hash function $\mathcal{H}_2$ is treated as a random oracle.

We are given $\mathbb{GD}$ as the output of $\mathsf{BiGrGen}$, $(g, Y, \tilde{g}, \tilde{X}, \tilde{Y})$ and unlimited access to oracle $\mathcal{O}$ from our challenger and have to output a tuple $(m^*, s, s^{x+m^* \cdot y})$ such that $s \neq 1_{\mathbb{G}_1}$ and m^* was

not asked to $\mathcal{O}$. In the first part of the proof, we will describe how $\mathcal{S}_4$ interacts with $\mathcal{F}_{\mathsf{RS}}$ and handles the interaction with $\mathcal{Z}$ and the real-world adversary $\mathcal{A}$. In the second part we analyze $\mathcal{S}_4$
Simulator $\mathcal{S}_4$ works as $\mathcal{S}_1$, except in the following cases:

Calls to $\mathcal{F}_{\mathsf{RO}}$: $\mathcal{S}_4$'s answers are generated the same way as $\mathcal{S}_1$ does.

Calls to $\mathcal{F}_{\mathsf{CRS}}$: $\mathcal{S}_4$ runs $\mathsf{PD.KeyGen}(\mathbb{GD})$ to obtain $\mathsf{PD}.pk := (u, v)$ and $\mathsf{PD}.td := \mathrm{dlog}_u(v)$. The common reference string is set to ($\mathbb{GD}$, $\mathsf{PD}.pk$, $\mathcal{H}$, $\mathcal{H}_1$, $\mathcal{H}_2$) according to the definition of $\mathcal{F}_{\mathsf{CRS}}$ in $\mathcal{G}$.

Calls to $\mathcal{F}_{\mathsf{CA}}$: $\mathcal{S}_4$ works exactly as $\mathcal{S}_1$.

Calls to $\mathcal{H}_2$: $\mathcal{S}_4$ manages the list $L_{\mathcal{H}_2}$ to respond identically to repeated requests. When some x is queried for the first time ($\mathcal{H}_2(x)$ is called for some $x \in \{0,1\}^*$), $\mathcal{S}_4$ chooses $\alpha_x \twoheadleftarrow \mathbb{Z}_p$, computes $\tilde{g}_x := \tilde{g}^{\alpha_x}$, and stores $(x, \alpha_x, \tilde{g}_x)$ in $L_{\mathcal{H}_2}$. Finally, $\mathcal{S}_4$ hands $\tilde{g}_x$ to the caller, as it is also done for repeated queries $\mathcal{H}_2(x)$, i.e. $(x, \alpha_x, \tilde{g}_x) \in L_{\mathcal{H}_1}$. Note that $\tilde{g}$ is used here, which is given by the PS1-Problem instance.

Registry Key Generation: For an honest P_{IDM} $\mathcal{S}_4$ sets $\mathsf{PS}.pk := (\tilde{g}, \tilde{X}, \tilde{Y})$, runs the algorithm $\mathsf{CS.KeyGen}(\mathbb{GD})$ to obtain $\mathsf{CS}.pk$ and $\mathsf{CS}.sk$, sets $pp := (\mathsf{PS}.pk, \mathsf{CS}.pk)$ and outputs pp. A corrupted P_{IDM} is managed by adversary $\mathcal{A}$.

User Registration: For an honest party P_i, $\mathcal{S}_4$ works exactly as $\mathcal{S}_1$.

For an honest P_{IDM} interacting with an honest party P_i, $\mathcal{S}_4$ executes the operations defined in Protocol Π_{RS}, but instead of computing a signature (σ_1, σ_2) itself, $\mathcal{S}_4$ queries its oracle $\mathcal{O}(usk_i)$, with usk_i given by $\mathcal{F}_{\mathsf{CA}}$, to obtain a valid signature for the registrating party.

For an honest P_{IDM} interacting with a corrupted party P_i, $\mathcal{S}_4$ executes Protocol Π_{RS} up to the point where P_{IDM} has to generate a signature for P_i. Now we rewind the adversary $\mathcal{A}$ up to the point where it sent its first message (pp', R) in behalf of P_i and respond with a new random challenge $ch' \neq ch$ (the same technique is used in [BCPV13]). Now, we extract usk_i, query $\mathcal{O}(usk_i)$ to obtain a valid signature for party P_i, and finalize the interaction according to Protocol Π_{RS}.

Product Addition: For honest party P_i, $\mathcal{S}_4$ chooses $\beta_{i,prod}, \gamma_{i,prod} \twoheadleftarrow \mathbb{Z}_p$ and sets $\tilde{g}_{i,prod} := \mathcal{H}_2(i, prod) = \tilde{g}^{\alpha_{i,prod}}$, according to the random oracle $\mathcal{H}_2$, $\tilde{X}_{i,prod} := \tilde{X}^{\alpha_{i,prod} \cdot \beta_{i,prod}}$, $\tilde{Y}_{i,prod} := \tilde{Y}^{\alpha_{i,prod} \cdot \beta_{i,prod} \cdot \gamma_{i,prod}}$, where $\tilde{g}, \tilde{X}, \tilde{Y}$ are given by the PS1-Problem instance. Then, $\mathcal{S}_4$ generates the non-interactive zero-knowledge proof of knowledge and outputs $ppk_{i,prod}$ as defined in Protocol Π_{RS}.

Purchase: For an honest party P_i, $\mathcal{S}_4$ works exactly as $\mathcal{S}_1$.

For an honest party P_j interacting with an honest party P_i, $\mathcal{S}_4$ executes the operations defined in Protocol Π_{RS}, but instead of computing a signature $\sigma_{i,j,prod}$ itself, $\mathcal{S}_4$ queries its oracle $\mathcal{O}(\gamma_{j,prod} \cdot usk_i)$, with $\gamma_{j,prod}$ chosen during $\mathsf{NewProduct}$ and usk_i given by $\mathcal{F}_{\mathsf{CA}}$, to obtain a pair (σ_1', σ_2'). Then $\mathcal{S}_4$ sets $\sigma_{i,j,prod} := (\sigma_1', \sigma_2'^{\beta_{j,prod}})$ and finalizes Protocol Π_{RS}.

For an honest party P_j interacting with a corrupted party P_i, $\mathcal{S}_4$ executes Protocol Π_{RS} up to the point where P_j has to generate a signature for P_i. Now we rewind the adversary $\mathcal{A}$ up to the point where it sent its first message $(\mathit{prod}, \mathit{ppk}, R)$ in behalf of P_i and respond with new random challenge $\mathit{ch}' \neq \mathit{ch}$ (the same technique is used in [BCPV13]). Now, we extract usk_i, query $\mathcal{O}(\gamma_{j,\mathit{prod}} \cdot \mathit{usk}_i)$ to obtain a pair (σ'_1, σ'_2), set $\sigma_{i,j,\mathit{prod}} \coloneqq (\sigma'_1, {\sigma'_2}^{\beta_{j,\mathit{prod}}})$, and finalize the interaction according to Protocol Π_{RS}.

For all remaining protocols (VfyProd, Rate, Verify, Link, Open, OProof, Judge) $\mathcal{S}_4$ works exactly as $\mathcal{S}_1$.

Now we show how $\mathcal{S}_4$ can be used to find a solution to the given PS1-instance.

When $\mathcal{F}_{\mathsf{RS}}$ outputs `error` in VfyRtg.7 some registered party P_i generated a valid rating $\sigma = (T_1, T_2, T_3, T_4, T_5, \mathit{ch}, s)$ without purchasing the corresponding product. We now rewind the whole game $\mathcal{G}$ up to the point where the random oracle $\mathcal{F}_{\mathsf{RO}}$ output ch for the first time. In the rewound game, $\mathcal{S}_4$ lets $\mathcal{F}_{\mathsf{RO}}$ output a new value $\mathit{ch}' \neq \mathit{ch}$. Eventually, $\mathcal{S}_4$ obtains a second valid rating $\sigma' = (T_1, T_2, T_3, T_4, T_5, \mathit{ch}', s')$ and we can compute usk_i using the extractor of Lemma 3.3. Furthermore, since σ and σ' are valid, (T_3, T_4) must be a valid signature for message usk_i under the public key $(\tilde{g}_{j,\mathit{prod}}, \tilde{X}_{j,\mathit{prod}}, \tilde{Y}_{j,\mathit{prod}})$:

$$\begin{aligned}&\mathrm{e}(T_3, \tilde{X}_{j,\mathit{prod}} \cdot \tilde{Y}_{j,\mathit{prod}}^{\mathit{usk}_i}) = \mathrm{e}(T_3, \tilde{g}_{j,\mathit{prod}}^{x \cdot \beta_{j,\mathit{prod}}} \cdot \tilde{g}_{j,\mathit{prod}}^{y \cdot \beta_{j,\mathit{prod}} \cdot \gamma_{j,\mathit{prod}} \cdot \mathit{usk}_i}) = \mathrm{e}(T_4, \tilde{g}_{j,\mathit{prod}})\\ &\implies T_3^{x \cdot \beta_{j,\mathit{prod}} + y \cdot \beta_{j,\mathit{prod}} \cdot \gamma_{j,\mathit{prod}} \cdot \mathit{usk}_i} = T_4.\end{aligned}$$

And we can compute

$$T_4^{1/\beta_{j,\mathit{prod}}} = T_3^{x + y \cdot \gamma_{j,\mathit{prod}} \cdot \mathit{usk}_i},$$

which is a valid signature for message $m = \gamma_{j,\mathit{prod}} \cdot \mathit{usk}_i$. Since oracle $\mathcal{O}$ is only queried for usk_i during Register and was not called during Purchase, we can output $(m, T_3, T_4^{1/\beta_{j,\mathit{prod}}})$ as a solution to the given PS1-Problem instance. The probability that m was already queried is negligible, because all values γ_x are chosen uniformly and independently at random.

When $\mathcal{F}_{\mathsf{RS}}$ outputs `error` in VfyRtg.9, a valid rating $\sigma = (T_1, T_2, T_3, T_4, T_5, \mathit{ch}, s)$ could not be opened by the honest P_{IDM}, which means that the rating was generated in behalf of an unregistered party. Furthermore, we know that $T_5 = \mathcal{H}_1(j, \mathit{prod})^{\mathit{usk}}$ for some $\mathit{usk} \in \mathbb{Z}_p$ and (T_1, T_2) is a valid signature for the message usk under the public key $\mathsf{PS}.\mathit{pk} = (\tilde{g}, \tilde{X}, \tilde{Y})$, because σ is valid. As described above, we rewind the whole game $\mathcal{G}$ to extract usk. $\mathcal{S}_4$ queries $\mathcal{O}$ only during the Register-protocol and the Purchase-protocol. Since P_{IDM} cannot open the rating, $\mathcal{O}(\mathit{usk})$ was not queried in the Register-protocol. The probability that $\mathcal{S}_4$ queried $\mathcal{O}(\mathit{usk})$ in the Purchase-protocol, meaning $\mathit{usk} = \gamma_x \cdot \mathit{usk}_i$ for some γ_x and some usk_i, is negligible, because all values γ_x are chosen uniformly and independently at random. Hence, we can output (usk, T_1, T_2) as the solution to the given PS1-Problem instance.

When $\mathcal{F}_{\mathsf{RS}}$ outputs `error` in RebLDB.18, then there exist to many valid, but non-linkable ratings for the given product. Since two ratings $\sigma' = (T'_1, T'_2, T'_3, T'_4, T'_5, \mathit{ch}', s')$, $\sigma'' =$

$(T_1'', T_2'', T_3'', T_4'', T_5'', ch'', s'')$ are linkable, iff $T_5' = T_5''$, there must exist at least one rating $\sigma = (T_1, T_2, T_3, T_4, T_5, ch, s)$, where $T_5 = \mathcal{H}_1(j, \mathit{prod})^{\mathit{usk}}$ for some usk that was not extracted during the $\mathsf{Purchase}$-protocol. Rewinding the game $\mathcal{G}$ and extracting usk, we can output $(\gamma_{j,\mathit{prod}} \cdot \mathit{usk}, T_3, T_4^{1/\beta_{j,\mathit{prod}}})$ as the solution to the PS1-Problem instance, as described above.

All outputs from $\mathcal{S}_4$ are distributed identically to the outputs of $\mathcal{S}_1$, assuming patching the random oracles does not fail. As argued previously, this only happens with negligible probability. Hence, when $\mathcal{Z}$ can distinguish between the game $\mathcal{G}$ and HYBRID at the Steps VfyRtg.7/9 or RebLDB.18 we can solve the PS1-Problem with non-negligible probability. □

4 Considering Revocation and Adaptive Adversaries

Revocation: The opening-proof mechanism $\mathcal{F}_{\mathsf{RS}}$ provides is a revocation technique that rescinds anonymity of the author of a single rating. More extensive notions of revocation include, but are not limited to:

- Revoke a user completely: the user cannot purchase products anymore, all existing ratings become invalid, and all future ratings will be invalid.
- Revoke all existing ratings of a user while preserving the ability to rate.
- Preserve existing ratings of a user but prohibit future ratings.

Which revocation technique to use depends on the higher-level application. Because of that, we do not integrate revocation in the definition of $\mathcal{F}_{\mathsf{RS}}$.

Nevertheless, Protocol Π_{RS} can be easily extended to support verifier-local revocation, which revokes a user completely: to revoke the party P_i the System Manager P_{IDM}, or even P_i himself, publishes the value $\tilde{Y}_i$ as the users' revocation token rt_i on a revocation-list $\mathcal{RL}$. Then any verifier can check whether the author of a given rating $\sigma = (T_1, T_2, T_3, T_4, T_5, ch, s)$ is revoked by testing if the equation $\mathrm{e}(T_5, \tilde{Y}) = \mathrm{e}(\mathcal{H}_1(j, \mathit{prod}), rt)$ holds for any entry $rt \in \mathcal{RL}$. Analogously, during $\mathsf{Purchase}$-requests the product owner can test whether $\mathrm{e}(M_i, \tilde{Y}) = \mathrm{e}(g_1, rt)$ holds to detect a revoked user P_i. This revocation mechanism conflicts with our definition of anonymity and it is an open problem how to prove security when revocation is considered.

Adaptive Adversaries: Theorem 3.1 only claims security against static adversaries, because anonymity and linkability are conflicting security properties, which impede the construction of UC-secure protocols in the presence of adaptive adversaries. To illustrate that, consider the following scenario with an adaptive adversary $\mathcal{A}$:

An honest party P_i rates some product and becomes corrupted after outputting the rating. Then the adversary $\mathcal{A}$ generates a second rating for the same product.

According to the definition of $\mathcal{F}_{\mathsf{RS}}$, the two ratings must be linkable. In the real protocol this is true (because the same keys can be used), but in the ideal protocol the simulator $\mathcal{S}$ does not obtain any identifying information about P_i during a Rate-request and has to simulate a rating. Hence, with overwhelming probability the ratings are not linkable in the ideal protocol and it is easy to distinguish between the ideal and the real protocol. Therefore, it seems unlikely that $\mathcal{F}_{\mathsf{RS}}$ is UC-realizable in the presence of adaptive adversaries. This problem will be investigated in future research.

References

[ACBM08] Elli Androulaki, SeungGeol Choi, Steven M. Bellovin, and Tal Malkin. Reputation Systems for Anonymous Networks. In *Privacy Enhancing Technologies*, volume 5134 of *LNCS*, pages 202–218. Springer, 2008.

[ACHdM05] Giuseppe Ateniese, Jan Camenisch, Susan Hohenberger, and Breno de Medeiros. Practical group signatures without random oracles. Cryptology ePrint Archive, Report 2005/385, 2005. `http://eprint.iacr.org/2005/385`.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 41–55. Springer, Heidelberg, August 2004.

[BCPV13] Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. Analysis and improvement of Lindell's UC-secure commitment schemes. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 13*, volume 7954 of *LNCS*, pages 534–551. Springer, Heidelberg, June 2013.

[BJK15] Johannes Blömer, Jakob Juhnke, and Christina Kolb. Anonymous and publicly linkable reputation systems. In Rainer Böhme and Tatsuaki Okamoto, editors, *FC 2015*, volume 8975 of *LNCS*, pages 478–488. Springer, Heidelberg, January 2015.

[BMW03] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 614–629. Springer, Heidelberg, May 2003.

[BPS^{+}17] N Busom, Ronald Petrlic, Francesc Sebé, Christoph Sorge, and Magda Valls. A privacy-preserving reputation system with user rewards. *Journal of Network and Computer Applications*, 80, 2017.

[BR07] Mihir Bellare and Phillip Rogaway. Introduction to modern cryptography, chapter 7 (course notes), 2007.

[BSZ05] Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 136–153. Springer, Heidelberg, February 2005.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[Can04] Ran Canetti. Universally composable signature, certification, and authentication. In *CSFW-17*, page 219, 2004.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Heidelberg, August 2001.

[CL06] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 78–96. Springer, Heidelberg, August 2006.

[CS97] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 410–424. Springer, Heidelberg, August 1997.

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 13–25. Springer, Heidelberg, August 1998.

[CSK13] Sebastian Clauß, Stefan Schiffner, and Florian Kerschbaum. k-anonymous reputation. In Kefei Chen, Qi Xie, Weidong Qiu, Ninghui Li, and Wen-Guey Tzeng, editors, *ASIACCS 13*, pages 359–368. ACM Press, May 2013.

[Dam02] Ivan Damgårg. On σ-protocols, 2002.

[Del00] Chrysanthos Dellarocas. Immunizing Online Reputation Reporting Systems Against Unfair Ratings and Discriminatory Behavior. In *EC 2000*, pages 150–157. ACM, 2000.

[FS07] Eiichiro Fujisaki and Koutarou Suzuki. Traceable ring signature. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007*, volume 4450 of *LNCS*, pages 181–200. Springer, Heidelberg, April 2007.

[Gro04] Jens Groth. Evaluating security of voting schemes in the universal composability framework. In Markus Jakobsson, Moti Yung, and Jianying Zhou, editors, *ACNS 04*, volume 3089 of *LNCS*, pages 46–60. Springer, Heidelberg, June 2004.

[GSW10] Essam Ghadafi, Nigel P. Smart, and Bogdan Warinschi. Groth-Sahai proofs revisited. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 177–192. Springer, Heidelberg, May 2010.

[HBBS13] O. Hasan, L. Brunie, E. Bertino, and N. Shang. A decentralized privacy preserving reputation protocol for the malicious adversarial model. *IEEE Transactions on Information Forensics and Security*, 8(6):949–962, 6 2013.

[HMQ04] Dennis Hofheinz and Jörn Müller-Quade. Universally composable commitments using random oracles. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 58–76. Springer, Heidelberg, February 2004.

[HZNR09] Kevin Hoffman, David Zage, and Cristina Nita-Rotaru. A Survey of Attack and Defense Techniques for Reputation Systems. *ACM Computing Surveys*, 42:1–31, 2009.

[Ker09] Florian Kerschbaum. A Verifiable, Centralized, Coercion-free Reputation System. In *WPES 2009*, pages 61–70. ACM, 2009.

[Lin11] Yehuda Lindell. Highly-efficient universally-composable commitments based on the DDH assumption. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 446–466. Springer, Heidelberg, May 2011.

[PLS14] Ronald Petrlic, Sascha Lutters, and Christoph Sorge. Privacy-preserving reputation management. In *SAC 2014*, pages 1712–1718. ACM, 2014.

[PS16] David Pointcheval and Olivier Sanders. Short randomizable signatures. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 111–126. Springer, Heidelberg, February / March 2016.

[Ste06] Sandra Steinbrecher. Design Options for Privacy-Respecting Reputation Systems within Centralised Internet Communities. In *Security and Privacy in Dynamic Environments*, volume 201 of *IFIP*, pages 123–134. Springer, 2006.

[ZWC$^+$16] Ennan Zhai, David Isaac Wolinsky, Ruichuan Chen, Ewa Syta, Chao Teng, and Bryan Ford. Anonrep: Towards tracking-resistant anonymous reputation. In *NSDI*, pages 583–596, 2016.