Personal Cross-Platform Reputation[*]

Johannes Blömer and Nils Löken
Paderborn University, Paderborn, Germany,
{johannes.bloemer, nils.loeken}@uni-paderborn.de

Abstract

We propose a novel personal reputation system for cross-platform reputation. We observe that, in certain usage scenarios, e.g. crowd work, the rater anonymity property typically imposed on reputation systems is not necessary. Instead, we propose a relaxed notion of rater anonymity that is more applicable in the crowd work scenario. This allows us to construct a secure personal reputation system from simple cryptographic primitives.

1 Introduction

Crowd workers, who perform tasks of variable complexity in a range of fields, e.g. design things, develop software, visit places in their area to take photographs, or generally serve as a distributed workforce for solving simple tasks, often operate on multiple platforms when selling their services to requesters, who seek solutions to their tasks. Platforms serve the purpose of bringing crowd workers and requesters together, typically focussing on a particular type of task, i.e. there are platforms for designers, text creators, software developers, etc. These platforms organize their work force, handle workers' payments, and enable requesters to contact specific workers if necessary — this feature would be of particular interest for platforms specializing in creative tasks.

Typically, platforms operate reputation systems to gather and provide information on requester satisfaction, e.g. satisfaction with the solutions a crowd worker provided. Such information is then displayed to other potential requesters under the assumption that the potential customer or requester would be satisfied with buying from a particular crowd worker if previous

[*]This work was partially supported by the Ministry of Culture and Science of the German State of North Rhine-Westphalia within research program "Digital Future".

This is an author's version of the paper of the same title originally presented at the Security and Trust Management workshop, STM 2019. The conference version is available from Springer via `https://doi.org/10.1007/978-3-030-31511-5_9`.

requesters were satisfied [10]. Therefore, it is important for crowd workers to build and maintain a good reputation.

However, since reputation systems are presently operated by platforms, there is a legitimate threat of vendor lock-in, i.e. crowd workers cannot reasonably switch platforms without losing all of their reputation. On the other hand, crowd workers often operate on multiple platforms in order to compensate for low numbers of offered tasks on individual platforms. Workers operating on multiple platforms can be expected to have lower reputation scores than their counterparts only operating on a single platform — not because of lower total requester satisfaction, but because the scores are split and stored separately across platforms.

We propose a *personal reputation system* that enables crowd workers to maintain their reputation scores themselves, and without tying these scores to specific platforms. Thus, personal reputation systems mitigate vendor lock-in and negate the disadvantage of reputations scores distributed among various platforms.

Related work. Reputation systems, in general, have been studied extensively [1, 2, 3, 4, 8, 9, 12], and in multiple disciplines. These systems are often restricted to single platforms or systems (such as peer-to-peer systems), but cross-platform reputation has also been considered.

As an example, Grinshpoun *et al.* [7] propose CCR, a model for cross-community reputation. Essentially, CCR allows online forums and similar communities to import reputation scores from other communities and to provide other communities with such scores. CCR mainly focuses on how to translate reputation scores, e.g. from a three-star scale to a five-star scale, while also considering differences in communities. As an example, consider a discussion forum on vehicle engines. In such a community, reputation may represent expertise in the topic discussed. The forum may allow new registrants to import reputation scores from an automobile discussion forum, but even if both communities use the same scale for their reputation scores, the engine discussion forum may not adopt the other community's scores on a one-to-one basis, because the automobile forum covers different topics, and an expert in automobiles is not necessarily an expert in engines. Hence, CCR weights reputation during translation.

While CCR considers the cross-community/platform aspect of our work, reputation scores are still stored at some platform; our personal reputation system goes as step further and the rated subjects store their reputation scores. However, concepts of CCR are applicable to our system when it comes to the interpretation of reputation, as the reputation has been obtained via different platforms.

Pingel *et al.* [14] implement a cross-platform reputation system for online forums. Their system is claimed to achieve several notions of security, anonymity, etc. The system assumes a centralized, but not necessarily trusted, service collecting reputation information from and distributing it to multiple communities. Since the reputation service is not trusted, and to grant users control over their data, users are to store their reputation scores themselves.

A drawback of Pingel *et al.*'s solution is the use of a centralized service to aggregate reputation scores; in contrast, our personal reputation system does not rely on centralized infrastructure.

As we see from the previous examples, reputation can be managed and stored in different ways, e.g. by centralized servers for individual or multiple communities or by the rated entities. Dennis *et al.* [5] discuss some (de-)centralized approaches to reputation storage and management in peer-to-peer networks, and then present decentralized storage of reputation via distributed ledgers, i.e. blockchain. While Dennis *et al.* consider the drawbacks of their approach in terms of limited throughput, they fail to mention the significant ecological impact of their proposal, particularly if implemented using proof-of-work blockchains.

While we use technologies similar to blockchains, we put away with proof-of-work and other consensus mechanisms. Still, our personal reputation system features some of the drawbacks of blockchains, e.g. in bandwidth consumption.

Impact of usage scenarios on anonymity guarantees for raters. Reputation systems in the cryptographic literature typically provide rater anonymity unconditionally or at least as long as raters behave honestly. This level of rater anonymity is important, because it has been shown that raters are more honest in their feedback if they are anonymous, i.e. they do not have to fear backlashes from ratees.

However, in the crowd work scenario, and focussing particularly on creative tasks such as text creation, rated solutions are one of a kind products, tailored towards a single and specific requester/customer. Due to product specifics, the creator of the product can be assumed to know the requester, who is a future rater of the product's creator. Then, due to details of products mentioned in reviews, ratees are potentially able to connect a review they receive to one of their creations, and thus the rater. A ratee may also be able to use temporal closeness of a provided solution and a received rating to link a rating to a rater. On the other hand, in order for the ratee to improve their future work based on criticism expressed in reviews, the ratee should know what product the review refers to. In turn the ratee knows the rater. Both views imply that, in the crowd work scenario, it is unlikely and undesirable for raters to remain anonymous towards the rated entity.

On a related note, a rater generally cannot be anonymous towards a platform that has mediated between the rater and the ratee: the platform must know the rater and the ratee in order to pay the ratee for her solution and bill the rater in return. The platform can use details from observed solutions and reviews to link raters and reviews in the same manner ratees can. As a consequence, for our personal reputation system, we relax the notion of rater anonymity typically imposed on reputation systems. Instead, we aim for *rater anonymity towards the general public*, so, given a rating, the rater remains anonymous towards every party not involved in the transaction.

Our contribution. Our personal reputation system combines concepts from previous work, such as decentralized storage of reputation and some technologies also used in blockchains with novel approaches to security. Our personal reputation system provides rater anonymity towards the general public, i.e. everyone except the rater, ratee and the platform. Adopting this weakened, yet sensible, notion of anonymity, we construct our personal reputation system from simple building blocks. Particularly, our construction uses hash functions, signatures, and commitment schemes.

Paper organization. In Section 2, we present the formal definition of personal reputation systems, as well as the building blocks used in our construction of a personal reputation system. We present our personal reputation system in Section 3. Finally, in Section 4, we discuss our design choices of our model of personal reputation systems, as well as our construction, and its limitations.

2 Preliminaries and building blocks

In this section, we first present our definition and security notions of personal reputation systems. We then proceed to present the building blocks used in our construction of a personal reputation system.

2.1 Personal reputation systems

On a formal level, a personal reputation system is a collection of algorithms and interactive protocols between the various entities involved, particularly platforms, raters, and ratees. In our example scenario from the introduction, crowd workers would take on the roles of ratees, while requesters would serve as raters.

After the system and some parties have been initialized, the raters can receive platform–transaction references (PTRs). A PTR is issued by a platform to a rater, and certifies that the rater has bought a service from some ratee. Eventually, the ratee hands out a rating token to the rater. Using the PTR and the rating token, the rater can then submit her review. Afterwards, everyone can verify that the ratee has received a given rating, and that the rating originates from a rater who has bought a product or service from the ratee via a given platform.

Definition 1. *Formally, a* personal reputation system *consists of four probabilistic algorithms* ($\mathsf{GlobalSetup}$, $\mathsf{RateeSetup}$, $\mathsf{PlatformSetup}$, Verify) *and three protocols* ($\mathsf{IssuePTR}$, $\mathsf{IssueRT}$, Rate)*, and features five types of parties: a parameter generator, ratees, platforms, raters, and verifiers. An entity may play the roles of multiple types of parties.*

GlobalSetup *is executed by a parameter generator which takes in security parameter 1^Λ and outputs public parameters pp.*

RateeSetup *is executed by a ratee who takes pp as input and outputs a private key usk and corresponding verification key uvk.*

PlatformSetup *is executed by a platform which takes in pp and outputs a private key psk and corresponding verification key pvk.*

IssuePTR *is executed between a rater and a platform who take pp as input; the rater additionally takes in the platform's verification key pvk; the platform additionally takes in her secret key psk, a helper string tid,*[1] *and a ratee's verification key uvk. The rater outputs a PTR ptr.*

IssueRT *is executed between a rater and a ratee who take pp as input; the rater additionally takes in the ratee's verification key uvk and her PTR ptr; the ratee additionally takes in her secret key usk. The rater outputs a rating token rt.*

Rate *is executed between a rater and a ratee who take in pp; the rater additionally takes in the ratee's verification key uvk, her rating token rt, and her (unprocessed) rating $review$; the ratee additionally takes in her secret key usk. Both parties output the (processed) rating r.*

Verify *is executed by a verifier who takes in pp, a ratee's and platform's verification keys uvk and pvk, respectively, and a (processed) rating r; the verifier outputs* valid *or* invalid*.*

For every security parameter Λ, helper string tid, and unprocessed rating $review$, we require

$$\Pr\left[\begin{array}{r}
pp \leftarrow \mathsf{GlobalSetup}(1^\Lambda),\\
(usk, uvk) \leftarrow \mathsf{RateeSetup}(pp),\\
(psk, pvk) \leftarrow \mathsf{PlatformSetup}(pp),\\
(ptr|\perp) \leftarrow \mathsf{IssuePTR}(pp, pvk|pp, psk, tid, uvk),\\
(rt|\perp) \leftarrow \mathsf{IssueRT}(pp, uvk, ptr|pp, usk),\\
(r|r) \leftarrow \mathsf{Rate}(pp, uvk, rt, review|pp, usk) :\\
\mathsf{Verify}(pp, uvk, pvk, r) = \mathsf{valid}
\end{array}\right] \geq 1 - \mathsf{negl}(\Lambda),$$

[1] *tid* could be a transaction or billing number.

where the probability is over the random choices of the algorithms.

We stress that the non-global setups are independent of each other, so ratees can work with multiple platforms, and platforms can work with multiple ratees. We also point out that, although we have no enrollment operation for raters, raters establish a permanent identity with platforms. After all, platforms will naturally want to send invoices to raters for services provided by the platforms. However, these identities are not directly part of our system, but influence our system in the form of the helper strings *tid* used in PTR generation. PTRs are tied to a single transaction. Given a rating, a platform should be able to identify the rater that has created the rating using a rating token generated from the PTR. Then, the platform should be able to identify *tid* based on the PTR.

For typical reputation systems, a couple of security properties have been proposed. Although there is no agreement in the literature on the exact formulation of these security notions, some security concepts show up time and again. Those notions are:

- binding of ratings to transactions,
- prevention of self-rating,
- linking of multiple ratings by the same rater for the same transaction,
- authenticity and integrity protection for ratings,
- rater anonymity, and
- traceability of misbehaving raters.

See [3] for a discussion and definition of these notions in the context of reputation systems. As a side note, it has recently been found that linkablility and traceability do not necessarily imply that two linked ratings can be traced to the same rater [11], but this does not pose a problem in the recent formulation of reputation systems in the universal composability framework [2], and it also does not pose a problem in our particular construction.

Our security notions for personal reputation systems are very similar to the ones proposed for reputation systems. Therefore, and we discuss our construction with respect to these notions on an intuitive level; this is feasible due to the simplicity of our construction. However, we formalize the notions of authenticity and integrity protection, i.e. rating unforgeability, and rater anonymity, specifically towards the general public.

The notion of rating unforgeability requires ratees to be unable to change ratings or to compute ratings themselves, without involvement of another party. The notion is defined relative to security experiment $\mathrm{Exp}^{\mathrm{forge}}$ shown in Figure 1.

```
$\mathrm{Exp}^{\mathrm{forge}}_{\mathcal{A}}(\Lambda)$:
Setup: set $P \leftarrow \emptyset$, $R = \emptyset$, computes $params \leftarrow \mathsf{GlobalSetup}(1^{\Lambda})$, and give $params$ to $\mathcal{A}$.
Query: $\mathcal{A}$ adaptively queries oracles
  • $\mathcal{O}_{\mathrm{PCr}}() \rightarrow pvk$ for creating new platforms.
  • $\mathcal{O}_{\mathrm{RCr}}() \rightarrow id_R$ for creating new raters.
  • $\mathcal{O}_{\mathrm{PTR}}(pvk, id_R, uvk) \rightarrow tid$ for issuing PTRs to raters.
  • $\mathcal{O}_{\mathrm{RT}}(id_R) \rightarrow \{0,1\}$ for issuing rating tokens to raters.
  • $\mathcal{O}_{\mathrm{Rate}}(id_R, review) \rightarrow \{0,1\}$ for making raters rate a ratee.
Responses: Upon query
  • $\mathcal{O}_{\mathrm{PCr}}()$:
    – $(psk, pvk) \leftarrow \mathsf{PlatformSetup}(params)$
    – $P \leftarrow P \cup \{(psk, pvk)\}$
    – give $pvk$ to $\mathcal{A}$.
  • $\mathcal{O}_{\mathrm{RCr}}()$:
    – $id_R \leftarrow |R|$, $R \leftarrow R \cup \{(id_R, \emptyset)\}$
    – give $id_R$ to $\mathcal{A}$.
  • $\mathcal{O}_{\mathrm{PTR}}(pvk, id_R, uvk)$:
    – check $\exists (s, pvk) \in P$ for some $s$
    – check $\exists (id_R, X) \in R$ for some $X$
    – if checks succeed:
        ∗ $tid \leftarrow_{\$} \{0,1\}^{\Lambda}$
        ∗ $(ptr|\perp) \leftarrow \mathsf{IssuePTR}(params, pvk|params, s, tid, uvk)$ playing the rater and the platform
        ∗ if the protocol did not abort: $Y \leftarrow X \cup \{(tid, (uvk, pvk, ptr), \perp, 0, \perp)\}$
        ∗ else: $Y \leftarrow X \cup \{(tid, \perp, \mathsf{invalid}, 0, \perp)\}$
        ∗ $R \leftarrow (R \setminus \{(id_R, X)\}) \cup \{(id_R, Y)\}$
        ∗ give $tid$ to $\mathcal{A}$
    – otherwise: give $\perp$ to $\mathcal{A}$
  • $\mathcal{O}_{\mathrm{RT}}(id_R, tid)$:
    – check $\exists (id_R, X) \in R$ and $(tid, (u,v,p), \perp, 0, \perp) \in X$ for some $u$, $v$, and $p$
    – if checks succeed:
        ∗ $rt \leftarrow \mathsf{IssueRT}(params, u, p)$ playing the rater while $\mathcal{A}$ plays the ratee
        ∗ if the protocol did not abort: $Y \leftarrow (X \setminus \{(tid, (u,v,p), \perp, 0, \perp)\}) \cup \{(tid, (u,v,,p), rt, 0, \perp)\}$
        ∗ else: $Y \leftarrow (X \setminus \{(tid, (u,v,p),, \perp, 0, \perp)\}) \cup \{(tid, (u,,v,p), \mathsf{invalid}, 0, \perp)\}$
        ∗ $R \leftarrow (R \setminus \{(id_R, X)\}) \cup \{(id_R, Y)\}$
        ∗ give 1 to $\mathcal{A}$
    – otherwise: give 0 to $\mathcal{A}$
  • $\mathcal{O}_{\mathrm{Rate}}(id_R, tid, review)$:
    – check $\exists (id_R, X) \in R$ and $(tid, (u,v,p), r', 0, \perp) \in X$ for some $u$, $v$, and $p$
    – check $r' \neq \mathsf{invalid}$
    – if checks succeed:
        ∗ $r \leftarrow \mathsf{Rate}(params, u, r', review)$ playing the rater while $\mathcal{A}$ plays the ratee
        ∗ $Y \leftarrow (X \setminus \{(tid, (u,v,p), r', 0, \perp)\}) \cup \{(tid, (u,v,p), r', 1, r)\}$
        ∗ $R \leftarrow (R \setminus \{(id_R, X)\}) \cup \{(id_R, Y)\}$
        ∗ give 1 to $\mathcal{A}$
    – otherwise: give 0 to $\mathcal{A}$
Output: Eventually, $\mathcal{A}$ outputs a tuple $(uvk^*, r^*)$. If for any $(psk, pvk) \in P$ $\mathsf{Verify}(params, uvk, pvk, r) = \mathsf{valid}$, and for all tuples $(id_R, X) \in R$ and all $(tid, (uvk, pvk, p), \cdot, \cdot, r) \in R$ we have $r \neq r^*$, then experiment outputs 1. Otherwise, the experiment outputs 0.
```

Figure 1: The rating forgery experiment for personal reputation systems played with adversary $\mathcal{A}$

The experiment keeps track of honest platforms and raters via sets P and R, respectively. Set P contains the honest platforms' secret and verification keys. Set R contains for each honest rater an identifier, and over time, also transaction identifiers, PTRs and rating tokens issued to the respective rater, as well as the rater's submitted rating. The goal is for the adversary to come up with a rating that Verify declares valid under a given ratee's verification key and any of the honest platforms' verification keys, while the rating has not been submitted by any of the honest raters.

In experiment $\mathrm{Exp}^{\mathrm{forge}}$, all ratees, as well as malicious raters and platforms, are played by the adversary. The adversary can set up new honest platforms and raters by calling the relevant oracles. Other oracles can be called for making honest platforms issue PTRs to honest raters and to make honest raters request rating tokens from ratees or make them submit reviews. In the experiment, honest platforms and raters do not interact with corrupt raters and platforms, respectively.

Definition 2. *A personal reputation system is* secure against rating forgery *if for all probabilistic polynomial time adversaries $\mathcal{A}$ we have* $\Pr[\mathrm{Exp}_{\mathcal{A}}^{forge}(\Lambda) = 1] \leq \mathsf{negl}(\Lambda)$*, where the probability is taken over the random choices of the adversary and the experiment.*

We now turn toward the notion of rater anonymity. Based on our crowd work scenario, as described in the previous section, full rater anonymity is neither feasible nor desirable. Instead we settle for the weakened notion of rater anonymity towards the general public. The notion requires the general public to not learn who submitted a particular rating to a ratee, but the ratee and the platform that mediated the transaction between the rater and the ratee may learn the rater's identity. Rater anonymity towards the general public is still necessary as to protect trade secrets of the rater, the platform, and the ratee. These considerations leads us to adopt the notion of rater anonymity towards the general public as the anonymity notion of choice for our personal reputation system. The notion of rater anonymity towards the general public is formalized with respect to indistinguishability experiment $\mathrm{Exp}^{\mathrm{anon}}$ as shown in Figure 2.

In the experiment, as before, sets P and R are used to track honest platforms and raters played by the experiment. Additionally, the experiment plays an honest ratee identified by verification key uvk^*. In contrast to the unforgeability experiment, in the anonymity experiment honest entities can interact with corrupt entities. This is reflected in the various variants of oracles used for different constellations of honest and dishonest entities interacting. Particularly, oracles for issuing PTRs, for issuing rating tokens and for submitting reviews exist in multiple variants. Variant C denotes the variants featuring honest raters, but says nothing about the honesty of ratees and platforms. Variant H denotes the variants featuring an honest non-rater, i.e. ratee or platform. Finally, Variant D, only present in the oracle for issuing PTRs, features both, honest platforms and honest raters. The oracle variants not only differ in the honesty of parties, but, as a consequence, also in their input and output behavior.

Definition 3. *A personal reputation system provides* rater anonymity towards the general public *if for all probabilistic polynomial time adversaries $\mathcal{A}$ we have* $|\Pr[\mathrm{Exp}_{\mathcal{A}}^{anon}(\Lambda) = 1] - 1/2| \leq \mathsf{negl}(\Lambda)$*, where the probability is taken over the random choices of the adversary and the experiment.*

2.2 Building blocks

We now present the building blocks that we use in our construction of a personal reputation system: commitment schemes, signature schemes and

$\underline{\mathrm{Exp}^{\mathrm{anon}}_{\mathcal{A}}(\Lambda)}$:

Setup: set $P \leftarrow \emptyset$, $R = \emptyset$, computes $params \leftarrow \mathsf{GlobalSetup}(1^{\Lambda})$, $(usk^*, uvk^*) \leftarrow \mathsf{RateeSetup}(params)$, and give $params$ and uvk^* to $\mathcal{A}$.

Query I: $\mathcal{A}$ adaptively queries oracles

- $\mathcal{O}_{\mathrm{PCr}}() \to pvk$ for creating new platforms.
- $\mathcal{O}_{\mathrm{RCr}}() \to id_R$ for creating new raters.
- $\mathcal{O}_{\mathrm{PTR:C}}(pvk, uvk)$, $\mathcal{O}_{\mathrm{PTR:D}}(pvk, id_R, uvk) \to tid$ and $\mathcal{O}_{\mathrm{PTR:H}}(pvk, id_R, tid) \to \{0,1\}$ for issuing PTRs; Variant C: corrupt rater, honest platform; Variant D: honest rater and platform; Variant H: honest rater.
- $\mathcal{O}_{\mathrm{RT:C}}()$ and $\mathcal{O}_{\mathrm{RT:H}}(id_R, tid) \to \{0,1\}$ for issuing rating tokens; Variant C: corrupt rater, honest ratee; Variant H: honest rater.
- $\mathcal{O}_{\mathrm{Rate:C}}()$ and $\mathcal{O}_{\mathrm{Rate:H}}(id_R, tid, review) \to \{0,1\}$ for making raters rate a ratee; Variant C: corrupt rater, honest ratee; Variant H: honest rater.

Responses: Upon query

- $\mathcal{O}_{\mathrm{PCr}}()$:
  - $(psk, pvk) \leftarrow \mathsf{PlatformSetup}(params)$
  - $P \leftarrow P \cup \{(psk, pvk)\}$
  - return pvk
- $\mathcal{O}_{\mathrm{RCr}}()$:
  - $id_R \leftarrow |R|$, $R \leftarrow R \cup \{(id_R, \emptyset)\}$
  - return id_R
- $\mathcal{O}_{\mathrm{PTR:C}}(pvk, uvk)$:
  - $tid \leftarrow_\$ \{0,1\}^{\Lambda}$
  - $\bot \leftarrow \mathsf{IssuePTR}(params, s, tid, uvk)$ playing the platform; $\mathcal{A}$ plays the rater
- $\mathcal{O}_{\mathrm{PTR:D}}(pvk, id_R, uvk)$:
  - if $\exists (s, pvk) \in P$ for some s and $\exists (id_R, X) \in R$ for some X:
    - $tid \leftarrow_\$ \{0,1\}^{\Lambda}$
    - $(ptr|\bot) \leftarrow \mathsf{IssuePTR}(params, pvk|params, s, tid, uvk)$ playing the platform and the rater
    - if the protocol did not abort: $Y \leftarrow X \cup \{(tid, (uvk, pvk, ptr), \bot, 0, \bot)\}$
    - else: $Y \leftarrow X \cup \{(tid, \bot, \mathsf{invalid}, 0, \bot)\}$
    - $R \leftarrow (R \setminus \{(id_R, X)\}) \cup \{(id_R, Y)\}$
    - return tid
  - otherwise: return $\bot$
- $\mathcal{O}_{\mathrm{PTR:H}}(pvk, id_R, tid, uvk)$:
  - if $\exists (id_R, X) \in R$ for some X and $\forall (t, \cdot, \cdot, \cdot, \cdot) \in X : t \neq tid$:
    - $ptr \leftarrow \mathsf{IssuePTR}(params, pvk)$ playing the rater; $\mathcal{A}$ plays the platform
    - if the protocol did not abort: $Y \leftarrow X \cup \{(tid, (uvk, pvk, ptr), \bot, 0, \bot)\}$
    - else: $Y \leftarrow X \cup \{(tid, \bot, \mathsf{invalid}, 0, \bot)\}$
    - $R \leftarrow (R \setminus \{(id_R, X)\}) \cup \{(id_R, Y)\}$
    - return 1
  - otherwise: return 0
- $\mathcal{O}_{\mathrm{RT:C}}()$:
  - $\bot \leftarrow \mathsf{IssueRT}(params, usk^*)$ playing the ratee while $\mathcal{A}$ plays the rater
- $\mathcal{O}_{\mathrm{RT:H}}(id_R, tid)$:
  - if $\exists (id_R, X) \in R$ and $(tid, (u, v, p), \bot, 0, \bot) \in X$ for some u, v, and p:
    - $rt \leftarrow \mathsf{IssueRT}(params, u, p)$ playing the rater; if $u = uvk^*$, the experiment also plays the ratee on input $(params, usk^*)$, otherwise $\mathcal{A}$ plays the ratee
    - if the protocol did not abort: $Y \leftarrow (X \setminus \{(tid, (u, v, p), \bot, 0, \bot)\}) \cup \{(tid, (u, v, p), rt, 0, \bot)\}$
    - else: $Y \leftarrow (X \setminus \{(tid, (u, v, p), , \bot, 0, \bot)\}) \cup \{(tid, (u, , v, p), \mathsf{invalid}, 0, \bot)\}$
    - $R \leftarrow (R \setminus \{(id_R, X)\}) \cup \{(id_R, Y\}$
    - return 1
  - otherwise: return 0
- $\mathcal{O}_{\mathrm{Rate:C}}()$:
  - $r \leftarrow \mathsf{Rate}(params, usk^*)$ playing the ratee
- $\mathcal{O}_{\mathrm{Rate:H}}(id_R, tid, review)$:
  - if $\exists (id_R, X) \in R$ and $(tid, (u, v, p), r', 0, \bot) \in X$ for some u, v and p, and check $r' \neq \mathsf{invalid}$:
    - $r \leftarrow \mathsf{Rate}(params, u, r', review)$ playing the rater; if $u = uvk^*$, the experiment also plays the ratee on input $(params, usk^*)$, otherwise $\mathcal{A}$ plays the ratee
    - $Y \leftarrow (X \setminus \{(tid, (u, v, p), r', 0, \bot)\}) \cup \{(tid, (u, v, p), r', 1, r)\}$
    - $R \leftarrow (R \setminus \{(id_R, X)\}) \cup \{(id_R, Y)\}$
    - return 1
  - otherwise: return 0

Challenge: Eventually, $\mathcal{A}$ outputs a two rater identifiers id_0, id_1 such that $(id_0, X), (id_1, Y) \in R$ for some X, Y, platform verification key pvk such that $(psk, pvk) \in P$ for some psk, and unprocessed rating $review$. Pick $b \leftarrow_\$ \{0,1\}$, perform actions of oracle queries $tid \leftarrow \mathcal{O}_{\mathrm{PTR:D}}(pvk, id_b, uvk^*)$, $\mathcal{O}_{\mathrm{RT:H}}(id_b, tid)$ and $s \leftarrow \mathcal{O}_{\mathrm{Rate:H}}(id_b, tid, review)$. If $s = 0$, the experiment outputs -1 and aborts.

Query II: Same as Query I.

Output: Eventually, $\mathcal{A}$ outputs a bit b'. The experiment outputs 1 if $b = b'$ and 0 otherwise

Figure 2: The rater anonymity experiment played with adversary $\mathcal{A}$

hash functions. In addition to these building blocks, we briefly introduce their security notions.

A *commitment scheme* can be likened to a sealed envelope. It allows a person to commit to a value without publishing the value, and later on publish the value and convince others that the published value is the value the person originally committed to; c.f. Ch. 5.6.5 of [13]. Commitment schemes are binding, i.e. the value committed to cannot be changed, and hiding, i.e. the value remains unknown to everyone except the person who has committed to the value.

A *signature scheme* is the cryptographer's equivalent to a handwritten signature. The signature is supposed to identify the signer in a way that the signer cannot feasible deny to have created the signature. In order to prevent adversaries from simply copying signatures, the signature must also consider the message that is being signed, as to make clear that the signer indeed intended to sign the given message; c.f. Ch. 12 of [13]. Signatures must be existentially unforgeable under adaptively chosen message attacks, i.e. it is hard to compute a signature under a message without knowledge of the signer's secret key, even if many signatures for other messages under the signer's secret key are known.

A family of *hash functions* is a keyed function used to compute short fingerprints or digests (hashes) of long messages; c.f. Ch. 5 of [13]. These functions are collision resistant, i.e. it is hard to find two distinct messages that result in the same digest.

3 Construction

In this section, we first review the hash chain principle that we use to construct a personal reputation system. From hash chains and the building blocks presented in the previous section, we then construct our personal reputation system.

3.1 Hash chain principle

Our personal reputation system makes use of the hash chain principle. We use two types of hash chain entries, namely self-signed certificates and data blocks. Hash chain entries are tied together ("chained") using a hash function.

Self-signed certificates can only occur as the initial entry of a hash chain. They are 2-tuples that consist of a signature verification key and a signature on the key. We call a self-signed certificate *valid* if the signature is a valid signature on the verification key under the verification key.

Data blocks are 3-tuples consisting of data, a hash, and a signature. Data blocks cannot occur at the start of a hash chain. The hash contained in a data block is a hash of another hash chain entry, called the predecessor;

hence, the hash establishes a successor/predecessor relation between hash chain entries, and thus, the chain.

If the hash is generated using a collision resistant hash function family, for any given set of hash chain entries, the successor and predecessor relations partially order the set with overwhelming probability, where the probability is over the random choice of the hash function from its family.

Considering a partially ordered set of hash chain entries, we call a data block *valid* if (1) the block's signature is a valid signature on the block's data and predecessor hash, and (2) the signature verification key that makes the data block's signature valid also makes the data block's predecessor valid. We call a set of hash chain entries valid, if it is totally ordered by the predecessor relation, and every element from the set is valid. It is easy to see that a valid set of hash chain entries contains exactly one self-signed certificate.

In summary, a hash chain scheme consists of four probabilistic polynomial time algorithms Setup, $\mathsf{Initialize}$, Append, and $\mathsf{VerifyChain}$. Setup is a parameter generation algorithm that chooses a concrete hash function and publishes its choice. $\mathsf{Initialize}$ sets up a new instance of the hash chain by establishing its initial block. Append adds a new block to an existing hash chain. $\mathsf{VerifyChain}$ verifies an existing chain as described above.

The successor and predecessor relations on hash chain entries give rise to the notions of minimal and maximal blocks for a set of hash chain entries: the minimal block is the only block from the set that does not have a predecessor in the set, whereas the maximum block is the only block from the set that does not have a successor block in the set; we denote the minimal and maximal blocks of a set E of hash chain entries as $\min E$ and $\max E$, respectively. Hence, for valid hash chains, the minimal block is the self-signed certificate, and the maximal block is the block most recently appended to the chain. For practical purposes, we assume the set of hash chain entries to be stored as an ordered set (ordered according to the predecessor relation), so we do not need to sort entries in our algorithms.

We now construct a concrete hash chain scheme from a signature scheme $\Sigma = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$, and a collision resistant hash function family $\mathsf{H} = (\mathsf{KeyGen}, \mathsf{Eval})$. Our hash chains work as follows.

Setup(1^Λ)**:** let $hfk \leftarrow \mathsf{H.KeyGen}(1^\Lambda)$ and output hfk.

Initialize(hfk)**:** let $(sk, vk) \leftarrow \Sigma.\mathsf{KeyGen}(hfk)$, let $\sigma \leftarrow \Sigma.\mathsf{Sign}(sk, \mathsf{H.Eval}(hfk, vk))$, and output $(sk, E = \{(vk, \sigma)\})$ as the hash chains secret key and first block, i.e. self-signed certificate.

Append$(hfk, sk, data, E)$**:** let $e \leftarrow \max E$ be the latest hash chain entry, compute $h \leftarrow \mathsf{H.Eval}(hfk, \langle e, data\rangle)$, compute $\sigma \leftarrow \Sigma.\mathsf{Sign}(sk, h)$, and output $E = E \cup \{(data, h, \sigma)\}$ as the (extended) hash chain.

$\mathsf{VerifyChain}(hfk, E)$: let $c \leftarrow \min E$ be the oldest hash chain entry, and parse c as (v, s). If parsing fails or $\Sigma.\mathsf{Verify}(v, \mathsf{H.Eval}(hfk, v), s) \neq \mathsf{valid}$, output $\mathsf{invalid}$. Otherwise, for all $e \in E \backslash \{c\}$ in ascending order (according to the successor relation), parse e as (d, h, s), let $e' \in E$ be e's predecessor and check that $\mathsf{H.Eval}(hfk, \langle e', d \rangle) = h$ and $\Sigma.\mathsf{Verify}(v, h, s) = \mathsf{valid}$. If either check fails for any $e \in E \setminus \{c\}$, output $\mathsf{invalid}$; otherwise output valid.

Our construction of a personal reputation system uses the hash chain principle in a gray box manner, i.e. we rely on the primitive as described above, but use the keys computed during setup and initialization in other contexts, too. For example the hash function key is used for hash function evaluations not related to the hash chain. Similarly, the signing key sk may be used to sign messages unrelated to the hash chain.

We also do not have any particular security notions for hash chains that we rely on when proving our construction secure. It should be noted though, that hash chains are fork consistent and append-only authenticated data structures if they are built from a collision resistant hash function family [6].

3.2 Our personal reputation system

As described before, our personal reputation system is to enable the distributed storage of ratings, independent of platforms. As a means for ratees to maintain a reasonable degree of control over their data (e.g. in compliance with data privacy laws), we have ratees store the ratings they receive. However, we employ the hash chain principle and the other primitives presented in the previous section in order to prevent ratees from tampering with the ratings. Particularly, if ratees try to reject unfavorable reviews or delete old ratings, this can be detected.

In our personal reputation system, *every ratee operates her individual hash chain.* We call a ratee's hash chain her "E-Set," and every E-Set is complemented by an "R-Set". The E-Set is used to register events, such as the issuance of a rating token to a rater, or the receipt of a rating from a rater. In order to prevent a ratee from rejecting an unfavorable rating, a rater first commits to her review and sends the commitment to the ratee. The rater sends the actual rating, including a decommit value only after she has received confirmation that the commitment has been appended to the ratee's E-Set. The rating is then stored in an R-Set entry that corresponds to the commitment's E-Set entry.

The previously mentioned platform–transaction references (PTRs) that platforms hand out to raters are certificates on a one-time identity established by a rater with the platform. The PTR certifies that the rater has bought a service from a rater. The platform that was involved in the transaction is also mentioned in the certificate, and the PTR is tied to the specific transaction. We use the term "one-time" loosely, because the same identity is used for multiple publicly observable interactions between a rater and a ratee, i.e. issuance of a rating token and submission of a review, but the identity is one-time in the sense that all these interactions are tied to a single transaction on the platform, i.e. if a rater happens to buy a service from the same ratee twice, the rater would use different identities. Although then the ratee may know that she interacted with the same rater multiple times, the general public does not learn this fact.

A rating token in our system is a certificate on a rater's one-time identity issued by a ratee, together with the ratee's PTR. Rating tokens are issued in order to publicly register a transaction and the rater's transaction-specific one-time identity in the ratee's E-Set.

We now present our personal reputation system. Let $\mathsf{CS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify})$ be a commitment scheme, and let HC be the hash chain constructed as in Section 3.1 that uses a collision resistant hash function family $\mathsf{H} = (\mathsf{Setup}, \mathsf{Eval})$ and a signature scheme $\Sigma = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$. Our personal reputation system is as described below.

GlobalSetup(1^Λ)**:** compute $cpp \leftarrow \mathsf{CS.Setup}(1^\Lambda)$, $hfk \leftarrow \mathsf{HC.Setup}(1^\Lambda)$, and publish $pp = (cpp, hfk)$.

RateeSetup(pp)**:** let $(usk, E\text{-}Set_{uvk}) \leftarrow \mathsf{HC.Initialize}(hfk)$ (i.e. the ratee's signing key and her self-signed certificate for verification key uvk that corresponds to usk), let $R\text{-}Set_{uvk} \leftarrow \emptyset$, publish $(E\text{-}Set_{uvk}, R\text{-}Set_{uvk})$, and privately output usk.

PlatformSetup(pp)**:** compute $(psk, pvk) \leftarrow \Sigma.\mathsf{KeyGen}(1^\Lambda)$, publish pvk, and privately output psk.

IssuePTR$(pp, pvk|pp, psk, tid, uvk)$**:** The rater first computes $(rsk, rvk) \leftarrow \Sigma.\mathsf{KeyGen}(1^\Lambda)$, and then sends her one-time identity rvk to the platform.

The platform fetches the self-signed certificate $e_0 \leftarrow \min E\text{-}Set_{uvk}$ of ratee uvk's hash chain, computes signature $t \leftarrow \Sigma.\mathsf{Sign}(psk, \mathsf{H.Eval}(hfk, \langle tid, e_0, rvk\rangle))$, and sends $ptr' = (pvk, uvk, rvk, tid, t)$ to the rater.

The rater privately outputs $ptr = (rsk, ptr')$.

IssueRT$(pp, uvk, ptr|pp, usk)$**:** The rater sends rvk to the ratee.

The ratee sends $crt \leftarrow \Sigma.\mathsf{Sign}(usk, \mathsf{H.Eval}(hfk, rvk))$ to the rater.

The rater computes $rt \leftarrow (rsk, ptr', crt)$, and engages in an execution of protocol Rate with the ratee: the rater's input is $(pp, uvk, pvk, rt, \perp)$, i.e. an empty review; the ratee's input is (pp, usk). Finally, the rater outputs rt.

$\mathsf{Rate}(pp, uvk, rt, review)(pp, usk)$: The rater verifies ratee uvk's hash chain $E\text{-}Set_{uvk}$ via $\mathsf{HC.VerifyChain}$ and aborts the protocol if verification fails. Otherwise, the rater identifies the hash chain's most recent block $e \leftarrow \max E\text{-}Set_{uvk}$, computes $(c, d) \leftarrow \mathsf{CS.Commit}(cpp, review)$, computes $data \leftarrow (rvk, ptr', crt, c)$ (where crt and ptr' are from rt and ptr, respectively), computes $h \leftarrow \mathsf{H.Eval}(hfk, \langle e, data\rangle)$, computes $\sigma \leftarrow \Sigma.\mathsf{Sign}(rsk, h)$, and sends $(data, h, \sigma)$ to the ratee.

The ratee uses $data' \leftarrow (rvk, crt, ptr', c, h, \sigma)$ to obtain a new (publicly observable) E-Set entry via$\mathsf{HC.Append}(hfk, usk, data', E\text{-}Set_{uvk})$. The ratee sends her signature from the new E-Set entry to the rater.

The rater computes $r = (ptr', d, review)$, privately outputs r, and sends r to the ratee.

The ratee (re-)publishes $R\text{-}Set_{uvk} \leftarrow R\text{-}Set_{uvk} \cup \{r\}$, and privately outputs r.

$\mathsf{Verify}(pp, uvk, pvk, r)$: Fetch the self-signed certificate $e_0 \leftarrow \min E\text{-}Set_{uvk}$ of ratee uvk's hash chain, verify $E\text{-}Set_{uvk}$ via $\mathsf{HC.VerifyChain}$, check $r \in R\text{-}Set_{uvk}$, check $\Sigma.\mathsf{Verify}(pvk, \mathsf{H.Eval}(hfk, \langle tid, e_0, rvk\rangle), t)$ (where tid, rvk, and t are from ptr' from r), check that arguments uvk and pvk match components uvk and pvk from r's ptr', and check that at most two entries $((rvk, crt, p, c, h, \sigma), h', \sigma') \in E\text{-}Set_{uvk}$ satisfy $p = ptr'$. If either check fails, output $\mathsf{invalid}$. Otherwise, let $((rvk, crt, p, c, h, \sigma), h', \sigma')$ with $p = ptr'$ be maximal in $E\text{-}Set_{uvk}$ with respect to the successor relation. Check $\Sigma.\mathsf{Verify}(uvk, \mathsf{H.Eval}(hfk, rvk), crt) = \mathsf{valid}$, check $h = \mathsf{H.Eval}(hfk, \langle rvk, crt, ptr', c\rangle)$, check $\Sigma.\mathsf{Verify}(rvk, h, \sigma) = \mathsf{valid}$, and check $\mathsf{CS.Verify}(cpp, (review, d), c) = \mathsf{valid}$. If either check fails, output $\mathsf{invalid}$; otherwise output valid.

Of course, parties do not trust each other, and thus do not trust the data sent by other parties. Therefore, all received data is recomputed and signatures are verified at the receiving end of communication. We have omitted such checks and computations to not clutter our presentation.

Efficiency. Unfortunately, the simplicity of the primitives used in our construction necessitates the whole hash chain to be requested, transmitted, and verified whenever someone wants to rate a ratee or verify a rating. Thus, both computation and bandwidth usage of these operations linearly depends on the number of previously issued rating tokens and the number of ratings. The inefficiency is caused by the need to verify ratees' hash chains ($E\text{-}Set$s) as part of these operations.

Mitigating efficiency bottlenecks. It stands to reason that replacing the hash chain in our construction by a more efficient authenticated data structure, particularly one that is fork consistent, append-only, and has concise proofs, may make our construction more efficient. However, we expect that exchanging hash chains for a more efficient data structure does not provide significant new insights. We also expect that changes to our construction due to incorporating a different data structure would be relatively minor and would not affect our construction's security.

Security. We now consider the security of our personal reputation system. To that end, we first discuss our construction's security against rating forgery in a formal manner. Afterwards, we discuss our system's other security properties.

Theorem 4. *If HC is instantiated using a collision resistant hash function family H and a signature scheme Σ that is existentially unforgeable under adaptively chosen message attacks, and CS is a computationally binding commitment scheme, then our personal reputation system is secure against rating forgery.*

Proof. We remind the reader about the hash-then-sign principle. That is, hashing a message with a collision resistant hash function and then singing the hash using a signature scheme that is existentially unforgeable under adaptively chosen message attacks (euf-cma secure) results in an euf-cma secure signature. A similar result holds for combining a binding commitment scheme and an euf-cma secure signature scheme in a commit-then-sign fashion, resulting in an euf-cma secure signature. This can be proven by adapting the proof for the hash-then-sign case, exchanging collision resistance for the binding property.

We now prove that every adversary that breaks our personal reputation system's security against rating forgery must break the security of at least one of the underlying primitives. To that end, we split the set of successful adversaries (adversaries that make experiment $\mathrm{Exp}^{\mathrm{forge}}$ output 1 with non-negligible probability) into two categories, depending on what part of our reputation system is successfully attacked. Type A adversaries forge signatures under a platform's signature verification key. Type B adversaries forge signatures under a rater's signature verification key.

However, we first have to argue that every successful attack falls into one of these categories. For that, we observe that a successful adversary outputs a tuple (uvk^*, r^*), such that, for one of the honest platforms set up over the course of the experiment, the platform's signature verification key satisfies $\mathsf{Verify}(pp, uvk^*, pvk, r^*) = \mathsf{valid}$. Reviewing the rating verification algorithm of our scheme, we find that only two of the components considered by Verify are not under control of the adversary, namely, the tuple $ptr' = (pvk, rvk, tid, t)$, and the tuple $rgs = (crt, p, c, h, \sigma)$, where $p = ptr'$, ptr' is part of r^*, and $r^* \in R\text{-}Set_{uvk}$. All other components are controlled by the adversary, particularly the ratees' $E\text{-}Set_{uvk}$ and $R\text{-}Set_{uvk}$ and their properties, as well as the signatures σ' contained in hash chain entries.

The public part of the PTR, ptr', is, among other things, a certificate on a rater's signature verification key rvk, which is used in the verification of signature σ. The certificate property of ptr' is verified under verification key pvk. From the fact that rating verification evaluates to valid under an honest platform's verification key and the fact that in experiment $\mathrm{Exp}^{\mathrm{forge}}$ honest platforms only hand out PTRs to honest raters, we know that a successful adversary must have come up with an honest platform's PTR for a rater controlled by the adversary (type A adversary), or relies on a PTR given to an honest rater, but changes/replaces that rater's review (type B adversary).

In either case, the adversary has to forge a signature that is created in accordance with the provably secure commit-then-sign and hash-then-sign principles, using primitives that are assumed secure. □ □

Impracticability of rating removal. It is noteworthy that our notion of personal reputation systems, and particularly, our scheme, allow for the removal of ratings. In order to remove ratings, a ratee has to de-publish the ratings' $R\text{-}Set_{uvk}$ entries.

However, everyone can estimate whether a given ratee deletes ratings by comparing the cardinalities of the ratee's $R\text{-}Set_{uvk}$ and $E\text{-}Set_{uvk}$. These cardinalities should be roughly the same (allowing for transmission errors and connection time outs for ongoing rating procedures, etc.). Large discrepancies should raise alarms. In order to avoid discrepancies in set cardinalities, the $E\text{-}Set_{uvk}$ entry that corresponds to the deleted $R\text{-}Set_{uvk}$ entry needs to be deleted as well. However, deleting an entry from $E\text{-}Set_{uvk}$ comes with a host of problems on its own.

Deleting a single old entry from $E\text{-}Set_{uvk}$, i.e. any block other than the most recent one, can be detected by everyone, because the deletion of the single entry results in the new $E\text{-}Set_{uvk}$ not being totally ordered, so every attempt at verifying the hash chain $E\text{-}Set_{uvk}$ will fail. The most recent entry from $E\text{-}Set_{uvk}$ may be deleted, but the deletion can be detected by the rater that created the corresponding rating, and the rater can even prove that fact via the signature from the deleted hash chain block: receipt of that signature is a condition for the rater to submit her rating to the ratee. Of course, $E\text{-}Set_{uvk}$ can also be truncated, i.e. all of the most recent entries are deleted, in which case the previous detection and proof method apply to each of the deleted blocks individually, and thus increase the probability of detection.

If a rater detects that one of her ratings has been deleted, she can take appropriate action outside the scope of our scheme. Potential actions include reporting the ratee who has deleted the rating to the platform that participated in creating the PTR used in the rating process, as well as suing the ratee.

Of course, in order for raters to detect rating deletion, they have to perform their checks repeatedly. This requirement of having raters occasionally check for rating deletion may seem to put an unnecessary burden on raters, especially in comparison to currently deployed reputation systems. However, the time intervals in between two checks can increase over time. For example, intervals could be doubled after each successful check. This is because we can expect new ratings to be added to an $E\text{-}Set_{uvk}$. At the same time, assuming a rational ratee, the benefit of removing one old rating may be offset by the cost of removing all of the old rating's successors (and thus the increased probability of rating deletion).

Note that the above discussion on rating removal is independent of our use of hash chains and applies to replacements as well, because a malicious ratee may simply roll back her $E\text{-}Set_{uvk}$ to an earlier state. However, as long as the authenticated data structure used to instantiate $E\text{-}Set_{uvk}$ is fork consistent and append-only, the above detection method will apply.

Further security properties. Now that we have discussed our scheme's protection of rating authenticity and integrity, and the threat of rating removal, we have a look at the other security notions for (personal) reputation systems listed in Section 2.1, and argue on an intuitive level that our personal reputation system satisfies these notions.

Regarding *identity management,* we see that all major parties, i.e. raters, ratees, and platforms, involved in our personal reputation system have certified identities, multiple of them in the case of raters. Platform and ratee identities are established by their respective public signature verification keys that may or may not be certified by a certification authority as part of a public key infrastructure; however, such a structure is beyond the scope of our system. Raters' one-time identities are certified by platforms and ratees via their respective signatures in PTRs and rating tokens, as well as their agreement on these identities, i.e. raters' signature verification keys. In addition, as mentioned before, raters' permanent identities are established by platforms (e.g. for billing purposes) and tied to raters' one-time identities via the helper strings *tid* which are contained in the publicly observable parts of PTRs, i.e. component ptr'.

Regarding *rater anonymity,* we note that our construction withholds rater's permanent identities from the public. This can be seen immediately from the fact that there is no public establishment of permanent rater identities in our construction. Instead one-time identities are established, one identity per transaction. Assuming the helper strings *tid* contained in PTRs, and particularly their public component ptr' contained in ratings, do not provide any information on raters' permanent identities to the public, the one-time identities of raters are unlinkable by the public in an information theoretic sense, both among each other and to the raters' permanent identities.

Thus, in the Exp^{anon} experiment, no adversary can do better than guessing what rater has performed the challenge rating. This establishes the following theorem.

Theorem 5. *If helper strings tid contain no information on rater's permanent identities, our construction achieves rater anonymity towards the general public.*

Although the public is unable to link a rating to a rater's permanent identity, the same one-time identity is used for multiple interactions between raters and other parties involved in the same transaction. This is because one-time identities and transactions are tied together by PTRs, and the same PTR is used for all interactions that are part of the respective transaction. Thus, PTRs allow for linking multiple ratings for the same transaction. Due to the aforementioned necessity of platforms establishing a permanent identity of raters, even though outside the scope of our reputation system, misbehaving raters can be traced. Furthermore, all ratings for a transaction can be traced to the same permanent identity.

Prevention of self-rating is a consequence from our use of PTRs. Particularly, ratees cannot forge PTRs or copy a PTR from another transaction, because this would require forging either raters' or platforms' signatures. However, ratees can legitimately obtain a PTR by buying a product or service from themselves (via an honest platform), or by setting up a dishonest platform themselves. In reputation systems that do not require raters, ratees, and platforms to be disjoint sets and do not enforce the use of a single (permanent) identity for all roles, such attacks are always possible. Our personal reputation system is an example of such a system.

However, in many practical scenarios, platforms require payment for their services; typically a percentage of per-transaction payments. Thus, ratees buying from themselves via an honest platform involves costs to the ratee. If the costs to the ratee are higher than the benefits from a rating, self-rating via honest platforms can be mitigated, at least as far as rational ratees are concerned.

On the other hand, since our personal reputation system explicitly allows ratees to work with multiple platforms, and platforms will have different types of reviews, e.g. 5-star scale or free text reviews, there is a need for reputation evaluation functions that help with interpreting ratees' reputation. Such reputation evaluation functions not only consider the actual review for a transaction, but also what platform has brokered that transaction. Platforms that nobody has ever heard of, e.g. platforms set up by dishonest ratees, will have effectively no influence when evaluating reputation. Thus, by using appropriate reputation evaluation functions, the benefits of maliciously set up platforms can be minimized, preventing rational ratees from obtaining PTRs from dishonest platforms.

In this context, CCR [7], c.f. related work, comes in. CCR considers such things as the confidence in reputation from different communities/platforms and weighs them accordingly. It should also be noted that reputation evaluation functions are an active research topic in economics.

4 Discussion

From theory's point of view, one can ask whether the primitives we use in our construction of a personal reputation system are necessary, or whether weaker building blocks suffice. For example, the signature scheme may be replaced by a *one-time signature scheme.* This is applicable to raters' signatures, because each rater creates exactly two signatures (and could have two one-time keys certified by the platform and the ratee as part of the PTR and the rating token). In practice, however, one-time signature schemes are often less efficient than signature schemes.

Additionally, applying one-time signatures does not allow the system to *recover from concurrent rating,* i.e. if two raters rate the same ratee at the same time, and thus use the same hash chain block as a basis for their computation, only one of the ratings can occur in the hash chain, while the other one has to be discarded. Otherwise, a fork of the hash chain occurs or rating verification will eventually fail (particularly, verification of the rater's signature inside a hash chain entry will fail). Hence, in order to recover from concurrent rating, one of the raters has to perform the rating process a second time, but based on the hash chain entry generated from the other rater's rating. While this is feasible with signature schemes, one-time signature schemes do not allow this (or rather: do not give any security guarantees in this situation).

From both, a theoretical and a practical point of view, one may criticise that our security model only considers security in the presence of malicious or rational ratees, but does not consider attacks by malicious platforms or raters, as well as collusion attacks. See [5] for discussions of some attacks of these types.

From a practical point of view, it is questionable whether platforms may actually be willing to participate in a personal reputation system, because personal reputation systems prevent vendor lock-in. After all, vendor lock-in may be a desirable feature from the vendor's perspective. However, our construction aims at minimizing the platform's involvement in the system.

A point of criticism with our system is that ratees have to be online constantly to accept new ratings or serve requests for their hash chains (E-Sets) and rating sets (R-Sets). We expect that specialized services will emerge to take on the roles of ratees.

Despite the potential benefits of such services, our reputation system provides the option for ratees to operate their own instance of the system, and thus, as long as ratees hold a copy of the singing key used for their hash chain, our proposal prevents vendor lock-in with respect to the service. The option for ratees to run their own instance of the reputation system also has the strong potential to prevent the formation of a single point of attack.

Services would also be helpful in normalizing reviews made on different scales, i.e. services that provide reputation evaluation functions mentioned at the end of the previous section. This type of service could re-use concepts from CCR [7], c.f. related work.

References

[1] Androulaki, E., Choi, S.G., Bellovin, S.M., Malkin, T.: Reputation systems for anonymous networks. In: PETS 2008. LNCS, vol. 5134, pp. 202–218. Springer (2008)

[2] Blömer, J., Eidens, F., Juhnke, J.: Practical, anonymous, and publicly linkable universally-composable reputation systems. In: CT-RSA. LNCS, vol. 10808, pp. 470–490. Springer (2018)

[3] Blömer, J., Juhnke, J., Kolb, C.: Anonymous and publicly linkable reputation systems. In: FC 2015. LNCS, vol. 8975, pp. 478–488. Springer (2015)

[4] Clauß, S., Schiffner, S., Kerschbaum, F.: k-anonymous reputation. In: ASIA CCS '13. pp. 359–368. ACM (2013)

[5] Dennis, R., Owen, G.: Rep on the block: A next generation reputation system based on the blockchain. In: ICITST 2015. pp. 131–138. IEEE (2015)

[6] Feldman, A.J., Zeller, W.P., Freedman, M.J., Felten, E.W.: SPORC: group collaboration using untrusted cloud resources. In: OSDI 2010. pp. 337–350. USENIX Association (2010)

[7] Grinshpoun, T., Gal-Oz, N., Meisels, A., Gudes, E.: CCR: A model for sharing reputation knowledge across virtual communities. In: WI 2009. pp. 34–41. IEEE Computer Society (2009)

[8] Hendrikx, F., Bubendorfer, K., Chard, R.: Reputation systems: A survey and taxonomy. J. Parallel Distrib. Comput. 75, 184–197 (2015)

[9] Ismail, R., Jøsang, A.: The beta reputation system. In: 15th Bled eConference. p. 41 (2002)

[10] Jøsang, A., Golbeck, J.: Challenges for robust trust and reputation systems. In: SMT 2009. Elsevier (2009)

[11] Kaafarani, A.E., Katsumata, S., Solomon, R.: Anonymous reputation systems achieving full dynamicity from lattices. In: FC 2018 (2018), `https://fc18.ifca.ai/preproceedings/87.pdf`

[12] Kamvar, S.D., Schlosser, M.T., Garcia-Molina, H.: The eigentrust algorithm for reputation management in P2P networks. In: WWW 2003. pp. 640–651. ACM (2003)

[13] Katz, J., Lindell, Y.: Introduction to Modern Cryptography, Second Edition. CRC Press (2014)

[14] Pingel, F., Steinbrecher, S.: Multilateral secure cross-community reputation systems for internet communities. In: TrustBus 2008. LNCS, vol. 5185, pp. 69–78. Springer (2008)