Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

ClearChart: Ensuring integrity of consumer ratings in online marketplaces

Pedro Moreno-Sanchez[a,*], Uzair Mahmood[b], Aniket Kate[a]

[a] CERIAS, Purdue University, West Lafayette, 47907, USA
[b] IMPR-CS, Saarland University, Saarbruecken, 66123, Germany

ARTICLE INFO

Article history:
Received 29 September 2017
Revised 2 April 2018
Accepted 24 April 2018
Available online 20 June 2018

Keywords:
Integrity
Recommendation systems
Online marketplaces
Distributed homomorphic MAC
Verifiable computation

ABSTRACT

Popular online marketplaces make an extensive use of ratings to inform their prospective buyers about best-rated products in their service. Given a strong inclination among online buyers towards buying the best-rated products, there is a clear monetary incentive to sellers, and in turn to service providers, to *unfairly* push their favored products at the top of the ratings lists. Due to the centralized nature of these systems, the problem is particularly hard to solve against undetectable attacks by service providers.

In this paper, we propose ClearChart, a transparency-enhancing mechanism to discourage this misbehavior in today's centralized marketplaces. Our protocol employs a novel distributed version of homomorphic MAC along with cryptographic accumulators and digital signatures to protect integrity of the ratings and improves verifiability of the ratings list. ClearChart introduces only a minimal storage overhead to the buyers and sellers, and can also tolerate collusion among sellers, the service provider and a subset of buyers. We have implemented ClearChart and demonstrated its practicality with an empirical evaluation.

1. Introduction

Online marketplaces such as eBay and Amazon heavily rely on ratings to help their customers navigate huge collections of products available through their services. Although advantages of ratings to users are unquestionable, prevalence of accumulated advantage phenomenon (or Matthew effect Times N.Y., 2007) in online marketplaces makes them highly susceptible to unfair practices: If low-ranked items were shown at the top of the ratings list, the eventual sales of these items might shoot up as they are shown to more users. The popularity of an item, and therefore its corresponding revenue, greatly depends on its rating.

As a result, service providers face attacks from malicious users aiming at biasing the ratings. For example, Lam and Riedl (2004) describe that there exist *eBay* users who tamper with ratings to boost their reputation. This raises an alarm about the potential consequences of tampering with recommendations and calls for a solution to ensure their integrity. Moreover, a malicious service provider might tamper with the rating of a product in a ratings list in return to a monetary reward. This active tampering may give undue advantage (or disadvantage) to the product in question. This becomes a particularly hard problem given the *undetectable* nature of the attacks supervised by *unaccountable* service providers.

One possible solution is to reduce the reliance on the service provider by having the sellers compute the ratings list themselves; however, among other issues, this can introduce a significant storage overhead to the sellers. Another possible solution is to distribute the service provider functionality among two or more non-colluding servers (Kerschbaum, 2009); however, instantiating such parties is not always possible for online marketplaces.

* Corresponding author.
E-mail addresses: pmorenos@purdue.edu (P. Moreno-Sanchez), umahmood@mpi-inf.de (U. Mahmood), aniket@purdue.edu (A. Kate).

https://doi.org/10.1016/j.cose.2018.04.014
0167-4048/© 2018 Elsevier Ltd. All rights reserved.

Our Contribution. In this work, we present ClearChart, a first line of defense against the active tampering of ratings lists in marketplaces. ClearChart allows verifiable computation of a ratings list in a marketplace, which can be checked by every party (including buyers), while introducing only a minimal storage overhead to the sellers.

We observe that a service provider generating a ratings list for its sellers is very similar to the scenario of verifiable computation on outsourced data (Backes et al., 2013). The ratings are forwarded (or outsourced) to the service provider, which performs a computation on them to generate the ratings list. This computation must then be verifiable by all other parties in the marketplace. Existing verifiable outsourcing solutions are designed for the single honest client scenario. However, in a marketplace there exist multiple parties outsourcing their ratings to the service provider and some of these parties might be malicious. To overcome this problem, we have defined a distributed version (DHMAC) of the homomorphic MAC primitive (Catalano and Fiore, 2013), which allows sellers to jointly verify the integrity of the computation for the ratings list done by the service provider. Only if sellers correctly verify the computation, they create a signed confirmation that convince buyers of the integrity of the ratings list. Moreover, only if service provider and sellers agree, a product can be entirely delisted from the marketplace.

We employ a novel combination of a token system based on RSA accumulators (Benaloh and De Mare, 1993) along with DHMAC to ensure integrity of the set of ratings. After a buyer submits a rating, sellers (as a whole) confirm its reception to the buyer via a signed confirmation message and to the service provider via an authentication tag produced by DHMAC. This ensures no rating can be dropped by the service provider. Moreover, only buyers with a token signed by the payment processor and present in the RSA accumulator can submit a legitimate rating. This ensures the validity of the submitted ratings.

Our experimental analysis shows that it is possible to carry out a transaction and leave a rating in less than 200 ms. The computation and verification of the ratings list takes less than 2 seconds for a (even unrealistically large) volume of 100,000 ratings. In general, our experiments show the feasibility of deploying ClearChart in practice.

2. Problem Definition

2.1. *Online marketplaces and the ratings*

Our simplified online marketplace composes of four main parties: *buyers* purchase products that are offered in the marketplace, *sellers* provide inventory to the marketplace, *service provider* is in charge of ensuring communication between buyer and seller associated to each transaction and finally the *payment processor* manages the payments associated to the transactions.

A transaction in such a marketplace is performed according to the following workflow. First, the service provider shows top rated products according to previous transactions. Given this concise information, the buyer decides which product to buy and routes the appropriate payment through the payment processor (e.g., *Visa* Credit Service). If the payment is correct, the corresponding seller receives the money and the product is shipped to the buyer. After a certain amount of time, the buyer might decide to leave a rating regarding the product. This rating must be then considered in future recommendations.

Each party in an online marketplace tries to maximize its utility. In particular, each buyer tries to get the best product for the lowest price. Each seller aims at maximizing its revenue by selling as many products as possible while contending for the recognition of this fact from the buyers. Finally, the objective of the service provider is to maximize its revenue through commissions on transactions by facilitating as many transactions as possible.

The ratings is the key element in a marketplace that enables to achieve such maximization goal. This is achieved by maintaining an updated ratings list, computed as a function (e.g., average) of the ratings associated to each of the carried out transactions. Buyers use the ratings list to select which product to buy next. Sellers verify using the ratings list that ratings regarding their products are being considered correctly. The service provider uses such ratings list as a proof to buyers and sellers that he is correctly running the online marketplace service. Therefore, ensuring the correct behavior of each of the parties regarding the ratings is vital for a sustainable online marketplace.

2.2. *Design goals*

The goal of the envisioned system to generate a ratings list for marketplaces is *integrity*. We follow the convention laid out in database systems (Motro, 1989) and divide the integrity goal into *validity* and *completeness*.

Validity Only a valid transaction must lead to a rating. Moreover, the service provider must not be able to tamper with an honestly generated rating value without being detected.

Completeness It must not be possible to drop any honestly generated rating associated to a valid transaction without being detected.

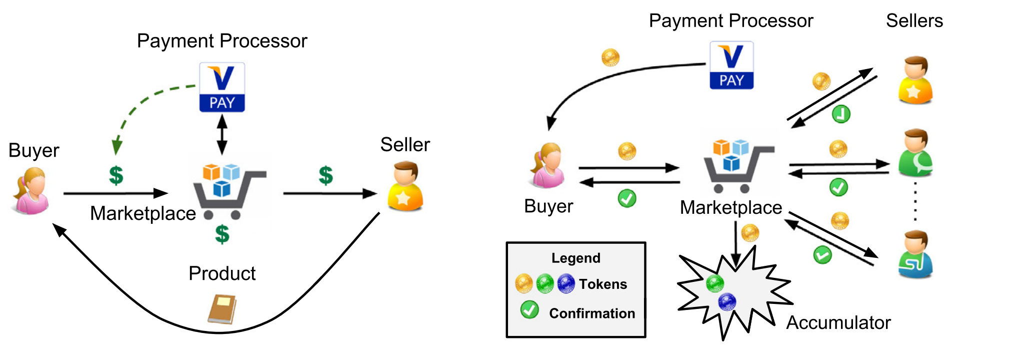
Besides integrity, a rating system should fulfill the following system requirements:

Efficiency The management of the ratings list should not impact the transaction and rating time.

Storage overhead Any party except from the service provider should need to store a reduced amount of data. The service provider will have considerable higher storage capacity and more data can be stored at its side.

2.3. *Threat model*

Given the undetectable nature of possible attacks from the service provider, we model the service provider as being malicious and assume that it can collude with the sellers. In addition, the service provider can create buyers on its own to further benefit itself and the sellers it is colluding with. We place no restrictions on the way that these malicious parties could



(a) Buyer performs a transaction and receives a product from Seller. (b) Buyer obtains a *token* from the payment processor and registers it with the service provider.

Fig. 1 – Transaction phase.

behave. We assume that the payment processor is honest. Additionally, we assume that the online sellers can communicate with each other through the service provider. We further assume that offline sellers verify the validity and completeness of the ratings list after coming online. As buyers do not know each other in advance, we assume that they need *not* interact with each other. We expect a majority of online sellers as being honest. Finally, we assume a public key infrastructure (PKI) to verify identities of the sellers, service provider and payment processor.

2.4. *Non-goals*

We strive to ensure integrity of ratings for a product sold by several sellers that compete to be at the top of the ratings list. If a product is sold solely by one seller, integrity of the ratings cannot be guaranteed; buyers anyways have no choice here other than buying the product from the seller.

We observe that a seller can introduce fake transactions (i.e., selling his products to himself) to falsely inflate his ratings. We do not plan to stop this inherent attack on every ratings system. Instead, we expect the system to ensure that a payment exists with the payment processor in this case such that carrying out this attack incurs some cost (e.g., payment fees) to the malicious seller.

While computing ratings list, we only consider linear operators. Modern recommender systems perform more complex calculation (e.g., standard deviation) over the ratings. We leave the management of complex ratings as an interesting future work. We consider ratings as a numerical value (e.g., an integer from 1 to 5) so that it is possible to rank different products. These are known as explicit ratings, and they are part of most of the current ratings (Wang and Tang, 2015). Integrating more elaborated ratings in ClearChart is also an interesting future work.

Finally, privacy preserving ratings have been proposed to ensure the privacy of the rating's provider, the rating itself or even both (Goodrich and Kerschbaum, 2011; Hasan et al., 2012). We focus on the integrity properties in this work and do not consider the privacy aspects.

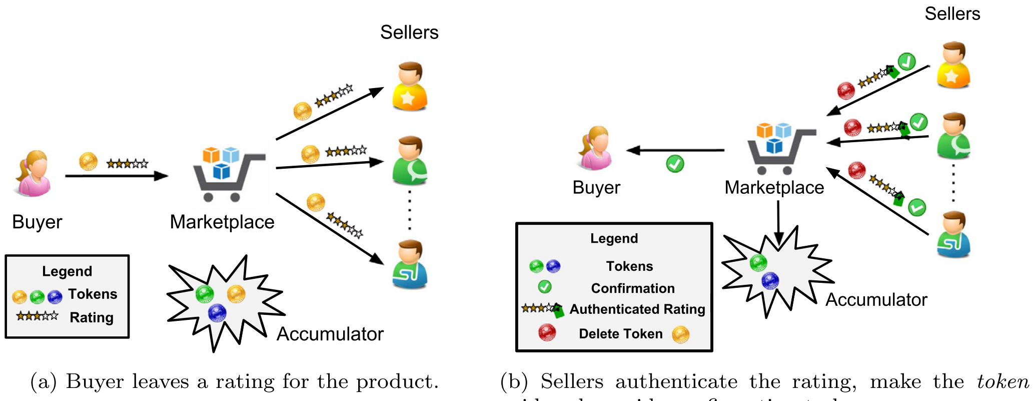
3. ClearChart protocol: Overview

Our ClearChart protocol can be roughly divided in three phases: the transaction phase, the rating phase and the computation and verification phase. For a single buyer, the transaction phase must be performed before the rating phase. Different buyers can perform these two phases independently. We assume that computation and verification are performed when enough ratings have been submitted (e.g., at the end of the day). If a misbehavior is detected, the protocol enters an additional blame phase. In the following, we sketch each of these phases.

Transaction phase In a nutshell, the buyer purchases a product and gets a token that can be used later to leave a rating, as depicted in Fig. 1. This token thus connects the transaction and rating phases for a single buyer. Note that such token is not present in current marketplaces and its adoption implies an update of their operational model. However, its integration enables integrity guarantees that can be interesting to their customers.

In detail, the buyer purchases a product from the ratings list shown to her. For that, the buyer performs a payment to the seller through the payment processor. Then, the seller ships the product to the buyer. Moreover, after checking the correctness of the payment, the payment processor provides the buyer with a token, which certifies the transaction information and the identity of the buyer (i.e., a fresh buyer's verification key of a digital signature scheme). Then, the buyer can register the token with the service provider and the sellers, who reply with a confirmation of its validity. This confirmation ensures that the service provider does not drop the token in the process since it cannot forge the confirmation on its own to fool the buyer. At this point, the buyer is eligible to leave a rating for the purchased product.

Rating phase The buyer leaves a rating for a previously purchased product, for what parties perform the steps depicted in Fig. 2. In detail, the buyer sends her rating and the token obtained in the previous phase to the service provider, along with a signature of this whole information using the



(a) Buyer leaves a rating for the product. (b) Sellers authenticate the rating, make the *token* void and provide confirmation to buyer.

Fig. 2 – Rating phase.

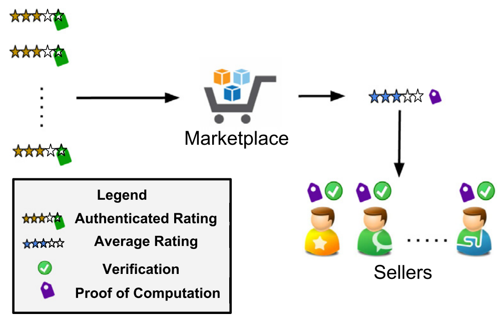


Fig. 3 – Computation and verification phase: Service provider uses the ratings and their authentication to produce an average rating for a product along with the proof. Sellers verify the proof.

signature key associated to her identity in the token. This signature avoids that the rating is forged by any party and ensures thus the authenticity of the individual ratings.

Then, the service provider forwards the buyer information to the sellers appending a proof of two facts: the token is associated to a valid transaction and the token cannot be reused to leave a rating. Sellers check the validity of the proof, authenticate the rating and keep track of the number of ratings seen so far for a given product. Interestingly, every seller authenticates the ratings for all the products, not only his own products. Finally, sellers send back to the service provider the authentication of the rating and a confirmation of the correctness of the whole process. Then service provider stores the authentication of the rating and forwards the confirmations to the buyer. These confirmations ensure that the rating has not been dropped by the service provider.

Computation and verification phase In this phase, the service provider creates a ratings list according to the ratings left by the buyers for each of the transactions. For that, the service provider computes a function of all received ratings for each of the products. We consider that service provider computes the average of the ratings, but other functions are also possible. Moreover, service provider uses authentications on the ratings provided by the sellers to compute a proof of correctness for the computation. The final ratings list and the proof are sent to the sellers who in turn verify the validity of the proof and that in fact all ratings have been considered in the creation of the ratings list. Intuitively, the sellers can perform this verification given that they know how many ratings have been left per product and those ratings have been authenticated by the sellers in the rating phase. Finally, the ratings list and an integrity confirmation issued by the sellers are made available to buyers. If buyers receive confirmations from enough sellers (e.g., majority of them), they start using the newly created ratings list for the subsequent purchases.

Blame phase If some party misbehaves during the protocol, a blame phase is entered to identify the misbehaving party. There are three cases to enter the blame phase due to a misbehaving seller or service provider. First, sellers cannot correctly verify a token created in the transaction phase. Second, sellers cannot correctly verify a rating and its associated token during the rating phase. Third, the proof of computation of the ratings list cannot be verified by the sellers. In all these cases, the sellers come together and use the message (e.g., token or proof of computation) to jointly identify the misbehaving party.

The blame phase is also entered due to a misbehaving buyer. There are two cases. First, the buyer uses a token not previously issued by the payment processor. Second, the buyer submits a rating using an invalid token. In both cases, the (cryptographic) information contained in the token can be used to blame the misbehaving buyer.

4. ClearChart protocol: Detailed construction

4.1. Building blocks

Digital signatures We use a digital signature scheme that must be existentially unforgeable under a chosen message attack. Given a message m and a signing key sk, we denote by $\sigma \leftarrow \mathsf{Sig}(sk, m)$ the signature of m using key sk. The verification algorithm $\mathsf{Verify}(vk, \sigma)$ outputs 1 if σ is a valid signature for m under the verification key vk. We instantiate this block with ECDSA (Johnson et al., 2001).

Accumulators Accumulators enable a succinct representation of a set of values. As a security property, we require that it is infeasible for an adversary to produce a proof of inclusion for an element not present in the accumulator. Function $\mathsf{Accumulate}(c, x)$ includes the element x in the accumulator c. Inversely, $\mathsf{Remove}(c, x)$ removes the element x from the accumulator c. Given the necessary auxiliary information aux [1], $\mathsf{GenMem}(aux, c, x)$ generates a proof w of $x \in c$. Finally, $\mathsf{CheckMem}(c, x, w)$ outputs 1 if w is a correct proof of $x \in c$. We instantiate this building block with RSA accumulators (Barić and Pfitzmann, 1997; Benaloh and De Mare, 1993).

Secret Sharing based MPC We use multi-party computation (MPC) for a set of functionalities that are realized in a (n, t) secret-sharing setting (Shamir, 1979). We require that less than $t + 1$ out of the n parties cannot reconstruct a secret $[s]$. $\mathsf{RandInt}()$ generates a secret share of a random value $[r]$. It can be instantiated using standard techniques. On input a message m, $\mathsf{PRFSS}(m)$ outputs a secret share $[\mathsf{PRF}(m)]$ (i.e., the result of applying a pseudo random function PRF on a message m). It can be instantiated from Cramer et al. (2005). The function $\mathsf{Inv}([r])$ outputs a secret share of $[r^{-1}]$. Finally, $\mathsf{EQ}([x], [y])$ returns a secret share $[b]$, containing 1 iff $x = y$. For them, we use the work by Aliasgari et al. (2013).

Distributed homomorphic MAC We have developed a novel distributed homomorphic MAC (DHMAC) inspired from the single client Homomorphic MAC construction by Catalano and Fiore (2013). In our scenario, we leverage the honest majority of sellers and we make a distributed version of it. We refer the reader to Appendix C for further motivation for the need of DHMAC.

DHMAC consists of the following functionalities. First, $[\sigma] \leftarrow \mathsf{Auth}(sk, \tau, m)$ performs the operations described in Algorithm 1. It takes as input the secret key, a label and a message and outputs a share of the authentication of the message. The reason for not allowing a seller to construct the whole tag in plain as $\sigma := (y_0, y_1)$ using the value r_τ is that it would allow a malicious seller to recover the secret key x.[2]

Algorithm 1: $[\sigma] \leftarrow \mathsf{Auth}(sk, \tau, m)$. /* compute a share of the authentication tag for m */.

```
Data: $sk := [x^{-1}]$, where $[x^{-1}]$ is the share of the inverse
      of $x \in \mathbb{F}_p$
      $\tau \in \{0,1\}^{poly(\kappa)}$ is the label associated to $m$
      and $m \in \mathbb{Z}_p$ is the message to authenticate
Result: $[\sigma]$ is a share of the authentication tag for $m$.
1 $[r_\tau] \leftarrow \mathsf{PRFSS}(\tau)$
2 $y_0 = m$
3 $[y_1] \leftarrow ([r_\tau] - m) * [x^{-1}]$
4 Output $[\sigma] \leftarrow (y_0, [y_1])$
```

Second, $\sigma' \leftarrow \mathsf{Eval}(f, \vec{\sigma})$ performs the same operations as in Catalano and Fiore (2013). Here, f is an arithmetic circuit defining the function to be computed over the outsourced data while $\vec{\sigma}$ is a vector of authentication tags created through Auth operation. The output of this algorithm is an homomorphic tag σ'.

Finally, $b \leftarrow \mathsf{Ver}(sk, z, P, \sigma')$ performs the operation described in Algorithm 2. The algorithm takes as input the secret key sk, the result of the computation z, $P := (f, \tau_1, \ldots, \tau_n)$ where f is circuit used in the computation and the labels (τ_i) correspond to the labels associated to the inputs of the computation, and the homomorphic tag σ' created through Eval. It is worthwhile to mention that in our specific case the degree of σ' is 1 as the circuit f consists of only additions and multiplication with a constant. Therefore, as in line 6 of Algorithm 2 these operations can be performed on the shares of the input giving rise to a share of the output[3]. It is also important to note that the construction in Catalano and Fiore (2013) suffers from a drawback where the tag size grows linearly with the degree of the arithmetic circuit. Since in our specific case, the degree of the arithmetic circuit is 1, this is not a problem. The output of this algorithm is the bit b set to 1 if z is the correctly verifiable result of the computation on the outsourced data.

Algorithm 2: $b \leftarrow \mathsf{Ver}(sk, m, P, \sigma')$ /* verify the validity of the function f computation */ .

```
Data: $sk := [x]$ where $[x]$ is the share of $x \in \mathbb{F}_p$,
      $m \in \mathbb{F}_p$ is the output of an evaluation circuit $f$,
      $P := (f, \tau_1, \ldots, \tau_n)$ where $f$ is the evaluation circuit
      and the labels ($\tau$'s) correspond to the inputs of the
      circuit,
      and $\sigma' := (y_0, y_1)$ is a homomorphic tag of degree 1,
Result: 1 or 0
1 if $y_0 \neq m$ then
2     Output 0
3 else
4     for $i \in \{1, \ldots, n\}$ do
5         $[r_{\tau_i}] \leftarrow \mathsf{PRFSS}(\tau_i)$
6     $[\rho] \leftarrow f([r_{\tau_1}], \ldots, [r_{\tau_n}])$
7     $[\rho'] \leftarrow y_0 + y_1 * [x]$
8     $[b] \leftarrow \mathsf{EQ}([\rho], [\rho'], \kappa)$
9     Output $b$ after reconstructing from $[b]$;
```

4.2. *The core protocol*

Setup We assume that a trusted party sets up the system at the beginning as follows. It randomly chooses a prime p and *exp-time*, a certain amount of preset time after which an operation is no longer considered valid by parties. Moreover, it initializes the accumulator $st_{mp} := \emptyset$ for the service provider and $st_{s_i} := \emptyset$ for each $seller_i$. Finally, it creates a different pair (sk, vk) of a digital signature scheme for the service provider and each of the sellers. Note that no key pairs are generated for buyers. Instead, each buyer will generate a pair for each transaction on the fly during the transaction.

After that, sellers interact with each other to finish their setup as follows. They compute $[x] \leftarrow \mathsf{RandInt}()$ and $[x^{-1}] \leftarrow$

[1] In our case, aux denote the factorization of the public modulus n associated to the accumulator.

[2] If the sellers know the value r_τ, the secret key x can be recovered as $(r_\tau - rating) * y_1^{-1}$

[3] This is described in further detail in Algorithm 4.2.

Inv([x]). Shared secret [x] severs as the secret key for the Ver algorithm in the distributed homomorphic MAC, while $[x^{-1}]$ is used as the secret key for the Auth algorithm. Finally, each seller locally initializes a pair $(p_{id}, \tau_{p_{id},s_i})$ for each product available in the marketplace. p_{id} identifies the product and $\tau_{p_{id},s_i} := (p_{id}, count := 0)$ denotes the label associated to p_{id} stored at s_i. *count* is a counter on the number of verified ratings associated to p_{id}. Finally, the service provider initializes its own pairs $(p_{id}, \tau_{p_{id},mp})$.

Transaction Assume *buyer*$_j$ wants to buy product p_{id} for a value v. Then, the transaction phase is executed in the following steps:

1. *Buyer*$_j$ generates a digital signature key pair $(sk_{b_j} := s, vk_{b_j} := g^s)$, where s is a number chosen uniformly at random and g is the generator of a group $\mathbb{G}_p$ of prime order p.
2. Since *buyer*$_j$ is not known ahead of time, the previous key pair needs to be certified. In order to do so *buyer*$_j$ routes through the payment processor a payment for a value v and additionally sends to the payment processor $tx := (p_{id}, vk_{b_j}, ts)$. Here, ts is a timestamp that expires after a time $ts + exp\text{-}time$.
3. The payment processor in response checks that the payment is correct and if so it generates $token \leftarrow \mathsf{Sig}(sk_{pp}, (tx, q))$, where q is a prime number chosen uniquely by the payment processor for each transaction. The payment processor provides *buyer*$_j$ with *tx, q* and *token*.
4. *Buyer*$_j$ forwards *tx, q* and *token* to the service provider. The service provider performs $st_{mp} \leftarrow Accumulate(st_{mp}, q)$, thereby adding q to the accumulator.[4] Then, the service provider forwards st_{mp}, q, *tx* and *token* to all sellers to get a confirmation from them and thereby convince *buyer*$_j$ that her *token* is indeed registered.

Algorithm 3: $b \leftarrow VerToken(st_{mp}, q, tx, token)$ /* verify the validity of *token* */.

```
Data: st_mp is the accumulator state of the service
      provider,
      q is a prime number,
      tx := (p_id, vk_bj, ts) where p_id is the product id, vk_bj is
      the verification key for buyer_j, ts is a timestamp,
      and token is a signature on (tx, q) by the payment
      processor.
Result: b A bit set to 1 if all verification succeed.
        Otherwise, b is set to 0
1 if $Verify(vk_{pp}, (token, (tx, q))$ and $time_{current} < ts + exp\text{-}time$
  and $Accumulate(st_{s_i}, q) = st_{mp}$ then
2     $st_{s_i} \leftarrow st_{mp}$
3     Output 1
4 Output 0
```

5. Each *seller*$_i$ invokes Algorithm 3. If this algorithm returns $\perp$, *seller*$_i$ initiates the *blame phase* using the condition which failed. Otherwise, *seller*$_i$ provides the service provider with its signature on q, $\sigma_{s_i} \leftarrow \mathsf{Sig}_{sk_{s_i}}(q)$.
6. The service provider collects all the signatures received from the sellers, bundles them as $(\sigma_{s_1}, ..., \sigma_{s_{|S|}})$ and forwards this to *buyer*$_j$ as a confirmation of her *token*'s registration.

Rating The rating phase is initiated when *buyer*$_j$ wants to leave a rating for a product (p_{id}), which she has previously purchased in $tx := (p_{id}, vk_{b_j}, ts)$, by redeeming the associated *token*. Following the convention of most online marketplaces, we assume that *rating* can take on values 1 to 5 inclusive, in increments of 1. The following takes place in the rating phase:

1. *Buyer*$_j$ picks a rating for p_{id} and creates a signature $\sigma_{b_j} \leftarrow \mathsf{Sig}(sk_{b_j}, (q, rating))$ where q is added to link the rating with *token* and rating is added in the signature to prevent any party from altering it. Then, *buyer*$_j$ forwards *token, tx, q, rating* and σ_{b_j} to the service provider.
2. In response, the service provider needs to assure the sellers that q is indeed in the accumulator. Only then, they authenticate the rating and then remove q from the accumulator. In order to do so the service provider creates a proof of membership $w \leftarrow \mathsf{GenMem}(aux, st_{mp}, q)$ and also computes the inverse of the prime number as q^{-1}. The service provider can calculate the inverse efficiently as it has access to the parameter *aux* of the accumulator. The service provider updates the label for p_{id} as $\tau_{p_{id},mp} \leftarrow (p_{id}, count + 1)$ and forwards $\tau_{p_{id},mp}$, *token*, tx, q, q^{-1}, w, *rating* and σ_{b_j} to the sellers.

Algorithm 4: $b \leftarrow VerReview(\tau_{p_{id},mp}, token, tx, q, q^{-1}, w, rating, \sigma_{b_j})$ /* verify the validity of the pair {*rating, token*} */.

```
Data: rating is the rating provided by the buyer,
      $\tau_{p_{id},mp}$ is the label for rating,
      token is a signature on tx and q and $\sigma_{b_j}$ is a
      signature on q and rating,
      q is a prime number and $q^{-1}$ is its inverse,
      and w is an accumulator witness of membership
Result: b A bit set to 1 if. all verification succeed.
        Otherwise, b is set to 0
1 if $Verify_{vk_{pp}}(token, (tx, q))$ and $Verify_{vk_{b_j}}(\sigma_{b_j}, (q, rating))$ and
  $\tau_{p_{id},s_i}.count + 1 = \tau_{p_{id},mp}.count$ and $CheckMem(st_{seller_i}, q, w)$
  and $tx.p_{id} = \tau_{p_{id},mp}.p_{id}$ and $isInverse(q, q^{-1})$ then
2     $st_{seller_i} \leftarrow Remove(st_{seller_i}, q, q^{-1})$
3     $\tau_{p_{id},s_i} \leftarrow (p_{id}, count + 1)$ /* store updated label */
4     Output 1
5 Output 0
```

3. Each *seller*$_i$ for $i \in \{1, ..., |S|\}$ invokes Algorithm 4. If the algorithm returns 0, *seller*$_i$ initiates the *blame phase* using the condition which fails. Otherwise, *seller*$_i$ generates its share of the authentication tag for the rating $[\sigma] \leftarrow \mathsf{Auth}([x^{-1}], rating, \tau_{p_{id},s_i})$ where $[\sigma] := (y_0, [y_1])$ as defined in DHMAC. The sellers gather together to reconstruct σ.
In addition *seller*$_i$ produces a signature on the rating, $\sigma'_{s_i} \leftarrow \mathsf{Sig}(sk_{s_i}, rating)$ which serves as confirmation for the *buyer*$_j$ of that her rating has been witnessed by *seller*$_i$. $\{\sigma'_{s_1}, \ldots, \sigma'_{s_{|S|}}\}$ and σ are forwarded to the service provider.

[4] It is worth noting that RSA accumulators require the accumulated values to be prime numbers, a restriction we fulfill in our protocol.

4. The service provider stores σ which is needed later to produce a proof of correct computation of the average rating and updates its accumulator, $st_{mp} \leftarrow \mathsf{Remove}(st_{mp}, q, q^{-1})$. Moreover, signatures $\{\sigma'_{s_1}, \ldots, \sigma'_{s_{|S|}}\}$ are forwarded to $buyer_j$ to assure her that the rating was correctly tallied by sellers.

Computation and Verification The computation phase takes place when it is time to compute a ratings list of top rated products, for example at the end of a day. In the following, we describe the steps performed to compute the average rating of a single product (e.g., p_{id}). Same steps are repeated for each of the products available in the marketplace.

1. The service provider computes the average of the ratings associated to p_{id}. For that purpose, the service provider creates an arithmetic circuit f of input size $n = \tau_{p_{id},mp}.count$, that computes the average of the inputs. Such arithmetic circuit can be realized as $(\mathrm{i}_1 + \ldots + \mathrm{i}_n) \cdot n^{-1}$, where i_j denotes the jth input. Then, the service provider computes the average rating for p_{id} as $z \leftarrow f(rating_1, \ldots, rating_n)$, using the n ratings associated to p_{id}.
2. The service provider computes a proof of correctness for the computation of f. For that, service provider first produces an homomorphic tag $\sigma' \leftarrow \mathsf{Eval}(ek := p, f, (\sigma_1, ..\sigma_n))$, using the evaluation function of DHMAC. Second, the service provider creates a signature $\sigma_{comp} \leftarrow \mathsf{Sig}(sk_{mp}, (z, \sigma'))$. Values z and σ' serve as proof of computation. Then, service provider sends σ_{comp}, z and σ' to the sellers. The non-repudiation property of the signature σ_{comp} prevents the service provider from denying that it signed the computation and its proof.
3. Each $seller_i$ verifies the proof received by the service provider using for that the Ver algorithm of the distributed homomorphic MAC (see Section 4.1). In detail, $seller_i$ calculates $n = \tau_{p_{id},s_i}.count$, reconstructs the circuit f and invokes $\mathsf{Ver}(sk := [x], z, P := (f, \tau_1, \ldots, \tau_{count}), \sigma')$. The set of labels are defined as $\{\tau_1 := (\tau_{p_{id},s_i}.p_{id}, 1), ..., \tau_{count} := (\tau_{p_{id},s_i}.p_{id}, \tau_{p_{id},s_i}.count)\}$ using the label τ_{p_{id},s_i} for p_{id} held by $seller_i$. Additionally, $seller_i$ checks the validity of the signature σ_{comp} by invoking $\mathsf{Verify}(vk_{mp}, (\sigma_{comp}, z, \sigma')$. If the homomorphic MAC verification algorithm or the signature verification returns 0, $seller_i$ can initiate the *blame phase* using the rejected proof or the failed signature. Otherwise, $seller_i$ can be assured that the average rating was computed correctly, and publishes a confirmation of the fact that she has correctly verified the computation. With enough of these confirmations (e.g., number of majority of the online sellers), buyers are convinced of the correctness of the generated ratings list.

Blame Phase This phase is reached whenever any of the sellers (e.g., $seller_i$) wants to create a proof of misbehavior to accuse the misbehaving party.

Each seller does three main verifications to detect a possible misbehavior: *token* verification, rating verification and the proof of computation verification. First, sellers must ensure during *token* verification that *token* has been correctly created by the payment processor, *token* has not expired and it has been correctly accumulated by the service provider. Second, sellers must additionally check during rating verification that the rating is correctly signed by the issuing buyer, the consistency of the label associated to the rating by the service provider and whether the *token* has been deregistered so that the rating cannot be replied. Finally, sellers must ensure that validity of the proof of computation created by the service provider after computing the required function over the outsourced data. Given the space constraints, we show the concrete set of verifications in Appendix B.

In case of a misbehavior is detected by $seller_i$, she bundles together the information that has failed to verify and forwards this information to all the other sellers in the marketplace. Each $seller_j$ different from $seller_i$ verifies the information that is sent forth by $seller_i$ and compares it with her own check. If the majority of the sellers agree that blaming information is correct then they blame the accused party. Otherwise, the sellers blame $seller_i$ for false accusation.

This phase can also be reached when a misbehaving buyer is detected. First, to detect an invalid token during the token registration, service provider and sellers must check whether $0 \leftarrow \mathsf{Verify}(vk_{pp}, (token, (tx, q))$ holds, where $token, (tx := (p_{id}, vk_{b_j}, ts), q)$ is provided by the buyer. In affirmative case, it is used as a proof of buyer misbehavior. Second, to detect an invalid token while leaving the rating, service provider and sellers must check whether $ts + exp - time$ has expired or whether $0 \leftarrow \mathsf{Verify}(vk_{b_j}, (q, rating))$, where $(q, rating)$ is provided by the buyer and vk_{b_j} must match the information in the token. Again, in affirmative case, it can be used as proof of misbehavior.

5. Performance analysis and system discussion

We have developed a prototypical Python implementation to demonstrate the feasibility of our construction. We have employed the MPC functionalities from an existing secret sharing-based MPC library (Smpc library, 2015). Our implementation comprises the transaction, rating and computation and verification phases.

Optimizations We have used several implementation-level optimizations in our implementation. The required operations for any two products are independent and thus can be parallelized. During the transaction and rating phases, sellers can run in parallel the verification checks defined in Algorithm 3 and Algorithm 4. Thus, we consider the work performed by each seller independently. In the computation and verification phase, sellers can create the tag associated to each label (Algorithm 2, lines 4–5) in parallel. Moreover, with the exception of step 8, the rest of steps of the algorithm can be performed locally by each seller independently from others. Finally, computation of the function f, the homomorphic tag σ' and its corresponding verification can be parallelized for each of the products in the marketplace. In this manner, an efficient implementation will take advantage of the multi-core architecture available in current computers.

Efficiency Operations for transaction and rating phases require only local operations for each party. Thus, we have tested these phases on a machine with an Intel Core i7 3.1 GHz processor and 16 GB RAM. First, the transaction phase takes approximately 112 ms from the moment the buyer decides to buy a product until she receives the confirmation from the sellers. Second, the rating phase takes approximately 80 ms from the moment the buyer decides to leave a rating, until she receives the confirmation from the sellers (i.e., including the time to perform the Auth over the rating by the sellers).

The computation and verification phase requires first the service provider to compute Eval. The service provider performs the operations from Eval over the plain values of the ratings and the authentications provided by the sellers. Therefore, service provider needs small time to perform such operations. As an example, we have obtained that service provider spends 52 ms to evaluate 100,000 ratings. Then, sellers need to interact to run the Ver algorithm. In our test, we use 7 different machines representing 7 different servers and assume that 3 of them are malicious. In that setting, we have obtained that 1.4 seconds are required to run the Ver algorithm using 100,000 ratings for a given product. As explained before, operations for different products can be parallelized and thus ClearChart scales with a growing number of products.

Interestingly, the computation and verification phase can be performed as a batch operation at the end of the day, when the system load is lower, thereby not interfering with the good performance of the other two phases.

Storage overhead The storage overhead property states that the sellers should need to store only a reduced amount of data. In ClearChart, each seller only needs to hold one label τ_{p_{id},s_i} per product id (p_{id}) irrespective of the number of purchases of that product. In addition, the sellers also store the accumulator as $state_{s_i}$ which is just a group element for an RSA based accumulator.

The verification for the average rating of a product in addition to the secret key, only requires the use of the sole label τ_{pid,s_i} that $seller_i$ holds for that pid. In this way it is guaranteed that the sellers do not store a lot of data and make use of outsourcing to verify computations over the ratings submitted by the buyers.

Usability A buyer needs to keep the token associated to a successful transaction to be able to give a rating later. A storage mechanism such as email could be used so that the token is available even when buyers log in through different platforms. The lightweight operations required in ClearChart for the buyers allow them to run the protocol even in somewhat resource constrained devices such as mobile phones: a buyer performs payments, forwards the tokens and verify confirmations (i.e., signatures over a message) from the sellers.

Although we focus on marketplaces, ClearChart can be applied to other scenarios where ratings are allowed after a certain product or service has been purchased, e.g., hotel booking sites such as booking.com.

6. Security analysis

Here, we show that ClearChart fulfills the validity and completeness goals.

Validity The validity goal states that only valid transactions must lead to a rating. Furthermore, the service provider must not be able to alter the ratings received from valid transactions without being detected. *Invalid* transactions here are the ones that have no payment associated with them and thus are not verified by the payment processor.

ClearChart meets validity as follows. First, ClearChart avoids *token* hijacking (i.e., stealing a token from an honest buyer) and replay attacks. We have elaborated these attacks in Appendix A. To prevent it, after a transaction the buyer gets a token from the payment processor and registers it with the service provider. The service provider must convince the sellers of the correct registration of the token who in turn send a confirmation to the buyer. In this manner, buyer makes sure token has been correctly registered at the service provider.

In detail, ClearChart prevent these attacks during the transaction phase given that $buyer_j$ gets a $token \leftarrow \mathsf{Sig}(sk_{pp}, (tx := (p_{id}, vk_{b_j}, ts), q))$ that certifies her verification key vk_{b_j}. Using it, further communication by $buyer_j$ can be verified. In particular, it allows to verify that the tx and q have not been tampered in its way to the sellers. The confirmations $(\{\sigma_{s_1}, \ldots, \sigma_{s_{|S|}}\})$ by the sellers assure that q has been correctly registered. Finally, the validity of the timestamp and the synchronization between st_{mp} and the individuals st_{s_i} ensures that only q has been accumulated and can be later used for leaving a rating.

Second, ClearChart is resilient to illegitimate ratings and binding attacks (e.g., leaving a rating for p_{id} using a token originally issued for p_{id}'). We elaborate this attack in Appendix A.3. To prevent it, the buyer submits a signed pair of rating and token. The signature ensures that rating is not changed by the service provider and the token ensures that rating is associated to a valid transaction. The sellers send a confirmation of these facts to the buyer so that any misbehavior from the service provider is caught.

In detail, ClearChart prevent these attacks during the rating phase given that $buyer_j$ and sellers can verify the correct communication of the rating, analogously to transaction phase. In this case, the confirmations $(\{\sigma'_{s_1}, \ldots, \sigma'_{s_{|S|}}\})$ by the sellers additionally assure that they have calculated the authentication tag σ on the correct rating. The honest majority of the sellers ensure that the authentication tag is correctly computed. The validity of the timestamp and the synchronization between st_{mp} and the individuals st_{s_i} ensures that rating is associated to a valid transaction and that only one rating is possible given a valid transaction. Finally, during the computation and verification phase, the service provider uses the correctly computed authentication tags $\vec{\sigma}$ to produce a proof of correct computation σ', which ensures that only correct ratings were utilized to produce the ratings list.

Completeness The completeness goal ensures that the service provider is not able to drop any of the ratings submitted by the buyers through valid transactions, without being detected. Intuitively, when the buyer submits a rating to the service provider, it must notify the sellers so that they can in turn confirm the reception to the buyer. Additionally, sellers authenticate the review using the DHMAC primitive. After computing the ratings list, the service provider outputs it along with a proof so that sellers are convinced that no valid rating has been dropped in the computation. In such case, a confirmation from sellers convince buyers of the correctness of the computed ratings list.

In detail, when a rating is received, the sellers check that the *count* in $\tau_{p_{id},mp}$ is consistent in Algorithm 4. Each $seller_i$ holds just one label τ_{p_{id},s_i} for each p_{id} and it checks that the incremented *count* in this label is equal to the one sent out by the service provider. This ensures that all the ratings are tallied correctly. During the verification of the average rating of a product, the sellers construct the circuit f that has *count* many inputs and the sellers make use of all the labels $(p_{id}, 1)$, ..., $(p_{id}, count)$. If the service provider has dropped any of the ratings, one of these labels would be missing in the computation; the verification would fail thus allowing the majority of sellers to detect the misbehavior, avoid to publish a confirmation for the new list and buyers to refuse the new list.

7. Related work

Several works (Aggarwal et al., 2018; PR, 2017; Zhuang et al., 2018) overview the importance of the rating system in online services such as marketplaces. Based on data from different online systems, these works show the impact of untrusted reviews and how they can lead the actions of other users in the system. For instance, in Aggarwal et al. (2018) authors identify Twitter users which manipulate their projected follower count to gain popularity among other users in the social network.

The problem of integrity in online systems is not new and there exist several works that deal with it. For example, *Frientegrity* (Feldman et al., 2012) is an approach to prevent an untrusted service provider from equivocating about the system's state. A honest user accepts the most recently seen state only when it is vouched by another user. They thereby leverage the ability of users to communicate to solve the problem. In a marketplace scenario, buyers are not known beforehand and therefore we cannot assume that they communicate with each other. Moreover, Frientegrity does not assure the correctness of the computation over the outsourced data.

Computing a ratings list in a verifiable manner is similar in nature to an e-voting scenario. In this sense, VoteBox (Sandler et al., 2008) provides a solution to ensure the correct outcome of an e-voting process. Such a scenario however is very restrictive in nature and allows for the computation of the best selling items only whereas in ClearChart, the service provider can generate any ratings list which associated function is representable as an arithmetic circuit. In addition, in e-voting there are only a limited number of candidates to vote for whereas in a marketplace the list of candidates (products) is significantly larger.

Our solution is motivated by the work of Backes et al. (2013). We model the computation of a ratings list as an instance of verifiable computation on outsourced data. One key difference between our work and that of Backes et al. is that in our model we have multiple clients in the form of sellers whereas they provide a solution for a single client scenario. This observation is not uncommon. Other solutions for verifiable computation like Parno et al. (2013) are also tailored for a single client. Introduction of multiple clients opens the door to collusion between the parties and this was one of the challenges for our work.

Kerschbaum (2009) proposes a system to deal with integrity of ratings, where rater and ratee may not know each other beforehand. In this system, the service provider is modeled as two non-colluding parties. We find such an assumption unrealistic in a marketplace. We do however make use of the payment processor, an honest entity that exists within our system and we provide a solution for a stronger adversarial model. On the other hand, we borrow from the solution of Kerschbaum by allowing only buyers that have purchased a product to leave a rating for that product and to allow only one rating per transaction.

More recently, Olga Ivanova (2017), propose a protocol that aggregate the ratings of a product in order to improve the integrity of the overall ratings. Although this approach reduces the performance costs of state-of-the-art aggregation methods, it affects negatively the rating of honest sellers as some of their ratings are not considered in the final published rating value. In ClearChart instead, all ratings are considered so that the ratings for honest sellers are not hampered.

The problem of accumulating *tokens* and providing one rating per transaction is similar to double spending. Nakamoto (2008) introduced a novel way to deal with it, where nodes maintain a ledger (the blockchain) and a coin is marked as being spent if it is included in the ledger. If a majority of nodes are honest, the problem of double spending is dealt with. It is possible then to apply this to our marketplace scenario with an honest majority of sellers. However, the sellers would have to store the entire ledger which fail to meet our storage requirements.

8. Conclusions

In this work, we present ClearChart, a system that uses a novel distributed verifiable computation over outsourced data protocol in conjunction with a token accumulator system to provide guarantees to buyers and sellers in a simplified marketplace ecosystem that the ratings list of top rated products has been generated with integrity; that is validity and completeness. We have envisioned a novel primitive, distributed homomorphic MAC, to allow multiple sellers outsourcing their data into the single service provider available.

We have implemented ClearChart and showed that a user spends less than 200 ms to perform a transaction and (possibly) leave a rating, therefore meeting the response time requirements for such an online system. The computation and verification takes less than 2 s for up to 100,000 ratings. Therefore, our study demonstrates the feasibility of deploying ClearChart in practice.

Finally, we do not claim ClearChart to be a silver bullet for all integrity issues with consumer ratings in online marketplaces. We propose these cryptographic solutions as a first line of defense against the undetectable rating manipulation in marketplaces, and expect them to be employed with other *system-level* measures to improve the integrity of online ratings.

Acknowledgments

We thank the anonymous reviewers for their feedback. This work has been supported by the German Universities Excellence Initiative.

Appendix A. Concrete security analysis

A1. Replay attacks

The buyer registers its *token* and gives its rating through the marketplace. Then replay attacks are possible given this communication between parties. The service provider mediates the communication of firstly, the *token* and secondly, the rating. The replay attack that is possible is that of multiple ratings that are left by the buyer and also the replay of a valid rating submitted by a buyer. In the following we discuss these attacks and how they are prevented in more detail:

Multiple Ratings Assume that a rating can be given without associated payment or a buyer can leave multiple ratings for a single payment, then the system can be flooded with ratings that may give unfair advantage to a seller. The attack scenario is as follows. The marketplace wants to create multiple ratings or create a buyer that purchases one product and submits multiple ratings such that the average rating for a malicious seller is achieved in the ratings list.

In order to prevent this attack, our protocol allows only *one rating per valid transaction*, where the valid transaction is one that involves an exchange of money witnessed by the payment processor. The payment processor provides only one unique *token* per transaction and this can be cashed in to provide a rating. This design prevents a malicious seller from gathering an unfair advantage as the only way a buyer can leave a rating on the part of the seller is if the buyer purchases a product. If the service provider wishes to facilitate the malicious seller in such a manner then the buyer created by the service provider will have to purchase the product for every rating it leaves and this is counted as a valid transaction for our system.

The mechanism to ensure *one rating per transaction* is provided by the signature of the payment processor (*token*), timestamp (*ts*) in the transaction ($tx := (p_{id}, vk_{b_j}, ts)$) and the use of the accumulator. The buyer has a limited window of time to obtain and register a *token* after a transaction because *ts* expires after time *exp-time*. *ts* therefore, needs to be valid for the amount of time it takes for this round of registration to happen as this ensures that only a single *token* is registered for this transaction. Later when a rating is submitted, the sellers remove this token from the accumulator (see Algorithm 4). The *token* can no longer be re-used as *ts* has expired and the payment processor will no longer provide a new *token* for the same transaction. This mechanism as a whole prevents multiple ratings for a transaction.

Replaying a Valid Rating If the service provider is able to replay a valid rating, it can give unfair advantage/disadvantage to a seller. The attack scenario is as follows. The service provider receives a good/bad rating from a $buyer_j$ who is part of a valid transaction. This rating is signed by the buyer using sk_{b_j} and the verification key vk_{b_j} is part of $tx := (p_{id}, vk_{b_j}, ts)$. The service provider can keep the *token* as being unused and replay the good/bad rating under the same signature to give unfair advantage/disadvantage to a seller.

Since each seller maintains the state of the accumulator, the *token* once used up is no longer a part of the accumulator and can no longer be added to the accumulator since *ts* has expired. This ensures that a rating submitted with a used *token* can no longer be replayed.

Since the sellers verify the accumulator state in Algorithm 3 and Algorithm 4 each, they ensure a consistent state of accumulator operations. In case of a discrepancy between the accumulator state of a seller and that of a service provider, the seller can invoke the blame phase (see Algorithm 5). In Algorithm 4 the sellers check that the *token* is valid and that the prime number q is part of the accumulator before removing it and providing authentication to the rating. This removal marks the *token* as being used and a re-addition of this *token* is prevented by *ts* in $tx := (p_{id}, vk_{b_j}, ts)$ and the signed *token* has expired. Therefore, it is not possible to submit the same rating again as the *token* for it has been used up.

Algorithm 5: *Blame* Identifies service provider misbehavior to be reported.

```
Data: Initiated by $seller_i$ for the phase in which it fails
Result: $blame\_info$ where this information is forwarded to
        other sellers to verify and label the accused party
1  if Algorithm 3 outputs $\perp$ then
2      if $0 \leftarrow Verify(vk_{pp}, (token, (tx, q))$ then
3          $blame\_info := token, (tx, q), service\ provider$
4      else if $time_{current} > ts + exp\text{-}time$ then
5          $blame\_info := ts, service\ provider$
6      else if $Accumulate(st_{s_i}, q) \neq st_{mp}$ then
7          $blame\_info := st_{s_i}, q, st_{mp}, service\ provider$
8  else if Algorithm 4 outputs 0 then
9      if $\perp \leftarrow Verify(vk_{pp}, (token, (tx, q))$ then
10         $blame\_info := token, (tx, q), service\ provider$
11     else if $0 \leftarrow Verify(vk_{b_j}, (\sigma_{b_j}, (q, rating))$ then
12         $blame\_info := \sigma_{b_j}, q, rating, service\ provider$
13     else if $\tau_{p_{id}, s_i}.count + 1 \neq \tau_{p_{id}, mp}.count$ then
14         $blame\_info :=$
           $\tau_{p_{id}, s_i}.count, \tau_{p_{id}, mp}.count, service\ provider$
15     else if $0 \leftarrow CheckMem(st_{seller_i}, q, w)$ then
16         $blame\_info := st_{seller_i}, q, w, service\ provider$
17     else if $tx.pid \mathrel{!}= \tau_{p_{id}, mp}.p_{id}$ then
18         $blame\_info := tx.p_{id}, \tau_{p_{id}, mp}.p_{id}, service\ provider$
19     else if $0 \leftarrow isInverse(q, q^{-1})$ then
20         $blame\_info := q, q^{-1}, service\ provider$
21 else if $Ver(sk := [x], z, P := (f, \tau_1, \ldots, \tau_{count}), \sigma')$ outputs 0
   then
22     $blame\_info := z, \sigma', service\ provider$
23 else if $0 \leftarrow Ver(vk_{marketplace}, (\sigma_{comp}, (z, \sigma'))$ then
24     $blame\_info := z, \sigma', \sigma_{comp}, service\ provider$
25 Output $blame\_info$
```

A2. *Token hijacking*

As part of our protocol the payment processor provides each *token* with a unique prime number. In this section, we show that it is possible to hijack a *token* if the same prime number is used more than once.

For the sake of simplicity lets assume that the same prime number q is part of two *tokens* $token := \mathsf{Sig}(sk_{pp}, (tx, q))$ and $token' := \mathsf{Sig}(sk_{pp}, (tx', q))$ where $tx := (p_{id}, vk_{b_j}, ts)$ and $tx := (p_{id}, vk'_{b_k}, ts')$. When both buyers register their *tokens* the state of the accumulator will be such that it has two entries for q. Lets say $buyer_k$ wants to hijack both *tokens*. It can then proceed as follows.

In the review phase of the protocol, the sellers only check to make sure that the prime number q that is part of *token* is present in the accumulator and that the signature on the rating provided verifies under the verification key vk_{b_j} that is part of $tx := (p_{id}, vk_{b_j}, ts)$ in *token*. $buyer_k$ can use *token′* twice to leave two ratings. The two ratings are signed by $buyer_k$ using the same signing key vk'_{b_k}. It is worthwhile to note that $buyer_k$ is entitled to leave the first rating. However, he succeeds in also hijacking the second token since q is still part of the accumulator after the first rating is submitted. In addition $buyer_k$ holds a valid *token*' with its own verification key and thus all the checks in Algorithm 4 succeed allowing $buyer_k$ to claim *token* as well.

A3. *Binding*

One of the properties that a rating in our protocol achieves is binding. A rating that is submitted during the rating phase is bound to the p_{id} for which it is intended. This means that it is not possible to use a rating submitted for a p_{id} in the computation of the average rating of a different p_{id}'.

The p_{id} that a buyer purchases is part of transaction *tx* and ultimately of a *token*. In Algorithm 4 the sellers check to see that the product id is the same in both *tx* and the label $\tau_{p_{id},mp}$ that is sent by the service provider. Having been assured of this each $seller_i$ for $i \in (1, ..., |S|)$ stores the updated label and generates $[\sigma] \leftarrow \mathsf{Auth}(sk := [x^{-1}], \tau_{p_{id},s_i}, rating)$. Notice, that the rating is now bound to p_{id} in τ_{p_{id},s_i} through $[\sigma]$. Since σ is used in the generation of a proof of correct computation and the verification algorithm makes use of the τ's; if the rating is associated with another product p_{id}' the verification of computation for both p_{id} and p_{id}' fails.

A4. *Conspiring sellers*

Imagine that a group of sellers are conspiring against a high-rated seller. Even in this case, ClearChart ensures the correctness of the ratings.

As discussed in the threat model (Section 2.3), the majority of sellers are assumed to be honest. Therefore, if there are n sellers, at most $n/2-1$ sellers are malicious. In this setting, at most $n/2-1$ sellers can deviate from the protocol and create up to $n/2-1$ malicious (and possibly distinct) messages $m^*_1, \ldots, m^*_{n/2-1}$. However, the majority of honest users would create at least $n/2$ *correct and equal* messages $m_1 = m_2 = \cdots = m_{n/2}$. In this scenario, the honest seller can always bundle together $m_1, \ldots, m_{n/2}, m^*_1, \ldots, m^*_{n/2-1}$ as a proof of two things: The correct message is m as the majority of sellers agreed on it, and the sellers with messages $m^*_1, \ldots, m^*_{n/2-1} \neq m$ were malicious.

Appendix B. Blame phase algorithm

Algorithm 5 defines the blame phase of the protocol. This algorithm is invoked by $seller_i$ when an inconsistency is detected in any of the checks. Algorithm 5 consists of three major portions, one each for the *token* verification, the rating verification and the proof of computation verification. In each of these parts $seller_i$ bundles together the information that has failed to verify and labels the service provider as the misbehaving party. She then forwards this information to all the other sellers in the set S. Each $seller_k$ for $k \in S$, $k \neq i$ verifies the information that is sent forth by $seller_i$ and compares it with its own check. If the majority of the sellers agree that the service provider is the misbehaving party then they blame the service provider otherwise, the sellers blame $seller_i$ for wrongly accusing the service provider.

Appendix C. Background on (Homomorphic) message authenticators

In this section, we provide the background of message authenticator schemes for a non-technical reader.

C1. *Message Authentication Codes (MAC)*

A *Message Authentication Code* (MAC) is used to authenticate a message. In practice, a MAC scheme is used by a pair of sender and receiver users to ensure that the message received by the receiver is an unmodified copy of the message intended by the sender.

A bit more technically, a message authentication code (MAC) is a tuple of algorithms $(\mathsf{KeyGen}, \mathsf{Auth}, \mathsf{Ver})$ defined as follows. $k \leftarrow \mathsf{KeyGen}(1^\lambda)$ is the key generation algorithm that takes as input the security parameter 1^λ and returns a symmetric key k. The authentication algorithm $\sigma \leftarrow \mathsf{Auth}(k, m)$ takes as input a key k and a message m and outputs a tag σ. Finally, the verification algorithm $b \leftarrow \mathsf{Ver}(k, \sigma, m)$ takes as input the key k, the tag σ and the message m and outputs 1 if σ is a valid tag for message m. Otherwise, it returns 0.

In practice, a MAC scheme can be used as follows. Assume two users Alice and Bob so that Alice has a message m and wants to send it to Bob. Further assume that both of them share a symmetric key k. Then, Alice first authenticates the message m by executing $\sigma \leftarrow \mathsf{Auth}(k, m)$, and sends the pair (m, σ) to Bob. When Bob receives such pair, he can verify that the message has not been modified with the verification algorithm by checking whether $\mathsf{Ver}(k, \sigma, m)$ returns 1.

Discussion The MAC scheme is based on shared symmetric-key between Alice and Bob and therefore it assumes that both have pre-agreed in a shared key k somehow. Moreover, this scheme misses the homomorphic properties that we require in ClearChart: Assume that Bob receives two pairs (σ_1, m_1) and (σ_2, m_2). Then, Bob cannot check whether $\sigma_1 + \sigma_2$ is a valid tag for the message $m_1 + m_2$. These two drawbacks are overcome with the definition of *homomorphic MAC schemes*.

C2. *Homomorphic Message Authenticator Codes (HMAC)*

An *Homomorphic Message Authenticator Code* (HMAC) allows the holder of a public evaluation key to perform computations over previously authenticated messages. This computation results on a single tag that certifies the correctness of the computation. In particular, a user knowing the corresponding secret key can verify that the tag authenticates the correct output of the computation.

In practice, a HMAC scheme can be used as follows. Assume two users Alice and Bob so that Alice contains a secret key *sk* and Bob has the corresponding public verification key *vk*. As in the standard MAC scheme, Alice can use *sk* to authenticate several messages $m_1, \ldots, m_n$, creating the corresponding tags $\sigma_1, \ldots, \sigma_n$. Assume now that Alice sends all this information to Bob. Bob can use his public verification key to compute $\sigma^* = \sigma_1 + \cdots + \sigma_n$ and $m^* := m_1, \ldots, m_n$. For simplicity we assume here the simple sum function, but many other functionalities are feasible in practice. Finally, Alice can verify that σ^* is a correct tag of message m^* and that it has been correctly computed as the sum of all the messages.

We refer the reader to Catalano and Fiore (2013) for a formal description and discussion of HMAC.

Discussion The HMAC cryptographic construction overcome the two challenges inherent to the standard MAC. First, Alice and Bob no longer share a symmetric key, but rather Alice can locally create a secret key and export the corresponding public key to Bob. Second, the homomorphic property of HMAC allows to perform computations over the authenticated data. In the example above, we refer to a simple sum function but more complex computations are possible with the constructions available in the literature.

On the other hand, HMAC is restricted to a single authenticator user (i.e., Alice) and a single verification user (i.e., Bob). Instead, in our protocol we require to have the same generic functionality as in HMAC but in the presence of multiple authenticator users (i.e., the sellers in our case). We overcome this challenge, by defining for the first time the *Distributed Homomorphic Message Authentication Code* (DHMAC). In a nutshell, the main difference resides on the fact that the secret key *sk* is now shared among a set of authenticators (i.e., the sellers) instead of a single user. Therefore, they need to collaborate in order to perform the authentication (i.e., creation of the tags $\sigma_1, \ldots, \sigma_n$) as well as the verification of the final outcome (i.e., m^*, σ^*). In practice, this allows to have a setting where the protocol can be correctly executed even in the presence of a few malicious participants. We refer the reader to Section 4.1 for the detailed description.

REFERENCES

Aggarwal A, Kumar S, Bhargava K, Kumaraguru P. The follower count fallacy: detecting twitter users with manipulated follower count 2018 arXiv:https://arxiv.org/pdf/1802.03625.pdf.

Aliasgari M, Blanton M, Zhang Y, Steele A. Secure computation on floating point numbers. Proceedings of NDSS, 2013.

Backes M, Fiore D, Reischuk RM. Verifiable delegation of computation on outsourced data. Proceedings of Computer & Communications Security; 2013. p. 863–74.

Barić N, Pfitzmann B. Collision-free accumulators and fail-stop signature schemes without trees. Proceedings of EUROCRYPT; 1997. p. 480–94.

Benaloh J, De Mare M. One-way accumulators: a decentralized alternative to digital signatures. Proceedings of EUROCRYPT; 1993. p. 274–85.

Catalano D, Fiore D. Practical homomorphic macs for arithmetic circuits. Proceedings of EUROCRYPT; 2013. p. 336–52.

Cramer R, Damgård I, Ishai Y. Share conversion, pseudorandom secret-sharing and applications to secure computation. Proceedings of Crypto; 2005. p. 342–62.

Feldman AJ, Blankstein A, Freedman MJ, Felten EW. Social Networking with Frientegrity: privacy and Integrity with an Untrusted Provider. Proceedings of USENIX Security; 2012. p. 31.

Goodrich MT, Kerschbaum F. Privacy-enhanced reputation-feedback methods to reduce feedback extortion in online auctions. Proceedings of CODASPY; 2011. p. 273–82.

Hasan O, Brunie L, Bertino E. Preserving privacy of feedback providers in decentralized reputation systems. Comput. Secur. 2012;31(7):816–26.

Johnson D, Menezes A, Vanstone S. The elliptic curve digital signature algorithm (ECDSA). Int J Inf Secur 2001;1(1): 36–63.

Kerschbaum F. A verifiable, centralized, coercion-free reputation system. Proceedings of workshop on privacy in the electronic society; 2009. p. 61–70.

Lam SK, Riedl J. Shilling recommender systems for fun and profit. Proceedings of WWW; 2004. p. 393–402.

Motro A. Integrity= validity+ completeness. ACM Trans Database Syst (TODS) 1989;14(4):480–502.

Nakamoto S. Bitcoin: a peer-to-peer electronic cash system. Consulted 2008;1(2012):28.

Olga Ivanova MS. How can online marketplaces reduce rating manipulation? A new approach on dynamic aggregation of online ratings. Proceedings of decision support systems; 2017. p. 64–78.

Parno B, Howell J, Gentry C, Raykova M. Pinocchio: nearly practical verifiable computation. Proceedings of IEEE S&P; 2013. p. 238–52.

PR R. A trustworthy reputation for online rating systems. Imp J Interdiscip Res 2017;3:1676–9.

Sandler D, Derr K, Wallach DS. Votebox: a tamper-evident, verifiable electronic voting system. Proceedings of USENIX security; 2008. p. 87.

Smpc library, 2015. https://freedom.cs.purdue.edu/software/ArithmeticsMPC_src_2014.12.09.tar.gz.

Shamir A. How to share a secret. Commun ACM 1979;22(11):612–13.

Times NY (2007). Is Justin Timberlake a product of cumulative advantage?http://www.nytimes.com/2007/04/15/magazine/15wwlnidealab.t.html?pagewanted=all&_r=1&. Accessed: 2015-06-05.

Wang J, Tang Q. Recommender systems and their security concerns. Cryptology ePrint Archive, Report 2015/1108, 2015. http://eprint.iacr.org/.

Zhuang M, Cui G, Peng L. Manufactured opinions: the effect of manipulating online product reviews. J Bus Res 2018;87: 24–35.

Pedro Moreno-Sanchez received his Ph.D. in the department of computer science at Purdue University in May 2018. He worked as a research assistant under the supervision of Prof. Aniket Kate. His doctoral research focused on the study and design of secure, privacy-preserving and decentralized credit networks. Pedro received a B.S. degree and M.S. degree in computer science from Universidad de Murcia in 2011 and 2013 respectively.

Uzair Mahmood is currently a software development engineer at Amazon. He received his M.S. degree in computer science from Saarland University in 2015 and his B.S. in electrical and electronics engineering from Lahore University of Management Sciences in 2012.

Aniket Kate is an assistant Professor in the computer science department at Purdue university. Before joining Purdue in 2015, he was a junior faculty member and an independent research group leader at Saarland University, Germany. He completed his postdoctoral fellowship at Max Planck Institute for Software Systems (MPI-SWS), Germany in 2012, and received his PhD from the University of Waterloo, Canada in 2010. His research integrates cryptography, distributed computing, and data-driven analysis to solve people-centric security/privacy problems in decentralized environments. For the last five years, he has been focusing on security, privacy and applicability of blockchain technology. For more details, visit the projects webpage: https://freedom.cs.purdue.edu/blockchains/