Beaver: A Decentralized Anonymous Marketplace with Secure Reputation

Kyle Soska*
CMU
ksoska@cmu.edu

Albert Kwon*
MIT
kwonal@mit.edu

Nicolas Christin
CMU
nicolasc@cmu.edu

Srinivas Devadas
MIT
devadas@mit.edu

***Abstract*—Amid growing concerns of government surveillance and corporate data sharing, web users increasingly demand tools for preserving their privacy without placing trust in a third party. Unfortunately, existing centralized reputation systems need to be trusted for either privacy, correctness, or both. Existing decentralized approaches, on the other hand, are either vulnerable to Sybil attacks, present inconsistent views of the network, or leak critical information about the actions of their users. In this paper, we present Beaver, a decentralized anonymous marketplace that is resistant against Sybil attacks on vendor reputation, while preserving user anonymity. Beaver allows its participants to enjoy open enrollment, and provides every user with the same global view of the reputation of other users through public ledger based consensus. Various cryptographic primitives allow Beaver to offer high levels of usability and practicality, along with strong anonymity guarantees. Operations such as creating a listing, purchasing an item, and leaving feedback take just milliseconds to compute and require generating just a few kilobytes of state while often constructing convenient anonymity sets of hundreds of transactions.**

I. Introduction

Reputation systems play a crucial role in establishing trust in online communities and drive many modern online businesses, ranging from auction markets to transportation companies. A typical reputation system features a collection of actors executing a protocol that allows users to leave reviews for their interactions with each other. Reviews, or feedback, usually consist of numeric ratings (e.g., 1–5 stars) and/or a short message. Feedback accumulates over time, and can be queried by other users in the system.

Many of the best known electronic commerce businesses (e.g., eBay, Amazon Marketplace, Uber, Airbnb, ...) operate as centralized marketplaces. Centralized marketplaces provide extremely weak privacy guarantees: Users need to trust the marketplace operator to maintain the correctness of the reputation state, and the confidentiality of sensitive information such as payment history (i.e., that it will not leak it or sell it to third-parties). Surveys of user sentiment indicate an increasing reluctance to putting such blind faith in commercial entities whose privacy policies have shown to be questionable [25].

An interesting new development in the realm of online reputation is that of online anonymous marketplaces [15], [41], frequently referred to as "darknet marketplaces." These marketplaces are built on the idea of anonymous commerce—they attempt to provide strong anonymity guarantees to buyers, sellers, and even marketplace operators. While online anonymous marketplaces have so far been primarily used for contraband and illicit items, a far more interesting point is that they strive to avail reputation systems with strong privacy and anonymity guarantees, and have proven to be economically viable. To achieve the desired anonymity properties, online anonymous marketplaces build on a combination of network-level anonymity—often running as Tor hidden services [20] or i2p "eep sites" [3]—and payment-level anonymity, using pseudonymous digital payment systems such as Bitcoin [36].

However, similar to "traditional" online marketplaces such as eBay, an online anonymous marketplace remains a centralized service. It thus needs to be trusted for availability as customers cannot query item listings or reviews if it is not online; it needs to be trusted for correctness, i.e., not inject fake reviews or suppress real ones; and it needs to be trusted not to link transaction history with private identifiers (e.g., shipping addresses communicated to vendors). Takedowns—e.g., of the Silk Road [15] or Silk Road 2.0 marketplaces—and the associated arrests of some of their patrons have evidenced that such centralized marketplaces often fail to provide the level of trust their users expect. Besides takedowns, "exit scams" frequently occur [41], where a marketplace unexpectedly closes, absconds with escrowed money—collected from customers, but not yet paid to vendors—and destroys all reputation information in the process. These shortcomings motivate the search for a solution that can provide strong anonymity without trusting a third party: in other words, a decentralized anonymous marketplace. The recently proposed OpenBazaar prototype [4] is one such distributed effort, but it currently does not provide strong anonymity properties. For instance, OpenBazaar relies on the UDP protocol and does not readily support network-level anonymization techniques such as Tor.

More fundamentally, decentralizing reputation systems has proven to be a challenging task. Early works (e.g., [17], [29]) present peer-to-peer/sensor network algorithms in which a node queries its peers to obtain the reputation for another node in the network. These approaches come with the drawback that each node's view of the network is biased by that of its peers. Another important challenge in decentralizing any reputation system, especially a system that protects users' anonymity, comes from the threat of Sybil attacks [21]. In a Sybil attack, an adversary creates a large number of identities in the network (customer accounts, nodes, etc.) and uses them to either inflate her own reputation or damage that of others. Intuitively, there seems to be a fundamental tension between the ability to identify a Sybil attack and the requirement that customers remain anonymous: *How can one be sure feedback is legitimate absent information about its source?*

In this paper, we introduce a formal model for a decentralized anonymous marketplace (DAM), and design Beaver, a

*Joint first authors.

Sybil-resistant DAM. Beaver is designed with e-commerce in mind, and consists of three types of participants: customers, vendors, and network miners. Unlike most existing approaches, participation in Beaver is free, open, and does not use a trusted third party. From the perspective of customers and vendors, Beaver behaves nearly identically to existing e-commerce systems such as Amazon Marketplace or eBay. It allows vendors to establish reputation by selling items (i.e., goods or services) to customers while ensuring that vendor reputation has not been adversely modified either positively or negatively. Beaver simultaneously provides strong anonymity to its customers, in that, unless the customer explicitly provides this information, no adversary can learn which purchases a customer has made or associate reviews with particular transactions.

Beaver builds on anonymous payment systems (e.g., Zerocash [6]), public ledger-based consensus protocols (e.g., Bitcoin "blockchain" [11]), and various cryptographic primitives to present a globally consistent view of the network to all of its users without sacrificing anonymity. Thanks to this consensus construction, Beaver is also able to avoid attacks where a few customers are targeted, and convinced of incorrect statements about another user's reputation.

All interactions in Beaver are performed via the consensus protocol. Concretely, item listings created by vendors as well as payments made or reviews left by a customer to a vendor are publicly available as part of the consensus. Customers can then freely and accurately enumerate all listings and feedback in the system, while deriving strong guarantees about the credibility of these reviews. Customers can also purchase products and leave their own reviews without fearing censorship or retribution. The major innovation in Beaver is that, although transactions and reviews are made public, the relationship between the transactions and reviews are kept private and the customers in Beaver always remain anonymous.

One of the key properties of Beaver is the mitigation of Sybil attacks. Traditional defenses against Sybil attacks rely on knowing the users' identities or their interaction history [35]. When the participants and their interactions are anonymous, as is the case with DAMs, such defenses cannot be deployed. Instead, we *anonymously* link reviews to transactions, by using non-interactive zero-knowledge proofs [10] and linkable ring signatures [31], which guarantee that there is a valid transaction for every review, and institute a small cost for each transaction. As a consequence, we can better understand and compute a notion we call *credibility*, corresponding to the lower bound on the cost to an adversary for generating feedback, and thus the trustworthiness of the current state of reputation. While Beaver is not Sybil-proof, we claim that it is Sybil-resistant under modest assumptions about the economic rationality of its participants.

We also evaluate Beaver by (1) measuring microbenchmarks on cryptographic primitives and common operations such as creating an item listing and generating and verifying an anonymous review, and (2) simulating Beaver operations using real online anonymous marketplace data gathered by Soska and Christin [41]. Our evaluation shows that most operations in Beaver take less than a second with only a few hundred bytes of space overhead to the ledger.

Shortly stated, we make the following contributions in this paper: (1) we formalize the requirements for a decentralized anonymous marketplace (DAM), (2) we design a Sybil-resistant DAM, Beaver, which makes use of novel applications of consensus, anonymous payments, zero-knowledge, and linkable ring signatures, (3) we analyze the security properties of Beaver. and (4) evaluate Beaver on a real marketplace data.

In §II, we discuss background into anonymous reputation systems and Sybil attacks on reputation systems. In §III, we formalize a model for decentralized anonymous marketplaces (DAM). We then introduce, in §IV, the high-level design of Beaver and some preliminaries for understanding the full system design. We delve in the details of Beaver in §V, perform a security analysis in §VI and a performance evaluation in §VII. Finally, we discuss some of our design choices in §VIII, related work in §IX, and conclude in §X.

II. Background

Reputation systems are used to collect, maintain, and distribute the performance scores of their users which can then be queried by other participants in the system. These scores can be used as a basis for establishing initial trust among users. Reputation plays an important role in online communities, especially in the context of e-commerce. In this setting, a customer assumes risk both when ordering goods or services whose quality she cannot verify ahead of time, and when purchasing from a vendor that she has no reason to trust.

A. Anonymous reputation systems

Customers leaving traceable public reviews leads to negative consequences; based on review history, adversaries (dishonest vendors, external observers, ...) can learn and use a considerable amount of potentially private information about the customers. For instance, Resnick and Zeckhauser [38] showed that vendors on eBay discriminate customers based on their review history. Moreover, if a vendor can associate reviews with transactions (which are often tied to sensitive information such as shipping addresses), then she may try to harm the customer in the future. This type of behavior has been observed on online anonymous marketplaces (see, e.g., [28]) and encourages customers to misreport their experiences in order to avoid harassment. To establish a fair market and a useful reviewing system, it is essential to ensure customers cannot be coerced into leaving inaccurate reviews. To achieve coercion resistance, we design a system where the reviews cannot be linked to particular transactions, and thus prevent adversaries from associating reviews with individual customers.

B. Decentralized reputation system

There have already been multiple proposals for anonymous reputation systems in the literature. While these systems provide anonymous reputation within their threat model, they rely on either a trusted third party [5], [7], [8], [19], [26], or require a trusted node among a set of powerful servers ("anytrust" model, [50]). Unfortunately, having a centralized network of a small number of nodes, even in the anytrust model, is undesirable as the adversary can perform targeted attacks. Moreover, in these systems, user enrollment is often expensive, as (semi-)centralized systems require the user or the trustees to perform an expensive operation to incorporate new members into the system.

On the contrary, modern cryptocurrencies like Bitcoin [36] support a fully decentralized model. That is, users are allowed

to join and leave the network at any time, and the security is guaranteed as long as some fraction of the network (rather than a small set of trustees) is honest. This makes it significantly harder for adversaries, especially those with limited resources, to undermine the security of the system. While Bitcoin and related cryptocurrencies do not support reputation or marketplace-specific features, we draw inspiration from the way that they handle decentralization.

C. Sybil attacks

Generally speaking, a Sybil attack [21] is an attack on distributed systems, where many nodes controlled by a few real entities cause the system to misbehave. In reputation systems, this attack concretely means that an adversary controls a large number of nodes and uses them to (1) generate positive reviews for herself to boost her reputation, and (2) leave negative reviews for others (e.g., her competition) to lower their reputation. Both scenarios are common in systems such as Yelp where users are free to review businesses without proving that they have ever been a customer. Mayzlin et al. [32] showed that allowing users to leave reviews without verifying that they have purchased the item generally leads to misbehavior. Similar attacks can also be found in systems like eBay or Yelp, where malicious vendors purposefully increase their ratings via numerous fake transactions with positive reviews or decrease other vendors' ratings.

Sybil attacks in practice are mitigated by having (1) a central authority (e.g., certificate authorities, Amazon, eBay, ...) verify the identities of the users making it harder to sign up (e.g., require a phone number for account creation), (2) making sure a review came from a valid transaction, and/or (3) analyzing behavior [47] to identify malicious users. Unfortunately, monitoring and limiting account creation contradicts our goals of open enrollment without a trusted third-party, and associating reviews with transactions or tracking behavior is at odds with providing customer anonymity.

A general defense against Sybil attacks that does not require a central authority is to increase the cost of adversarial actions in the system, thereby discouraging an economically rational adversary. This is commonly done via expensive operations such as proof-of-work [27], CAPTCHAs [45], or by adding a fee to reputation-generating actions. In this paper, we show that our system can charge fees (in a way that is particularly natural for marketplaces, and in many cases cheaper than existing marketplaces) to deter Sybil attacks, without sacrificing our goals of anonymity and decentralization.

III. Decentralized Anonymous Marketplaces

In this section, we present definitions and goals for a decentralized anonymous marketplace (DAM).

A. Definitions and notations

1) DAM Components: A DAM consists of different components that interact with each other through transactions, which are described here.

Customers. Customers in a DAM make purchases from vendors, and may leave reviews for their purchases.Let $\mathcal{C}$ be the set of honest customers $\{c\}$, and $\tilde{\mathcal{C}}$ be the set of malicious customers $\{\tilde{c}\}$.

Vendors. Vendors sell *items*, which can be any sort of goods or services of monetary value. Similar to customers, let $\mathcal{V}$ be the set of honest vendors $\{v\}$, and $\tilde{\mathcal{V}}$ be the set of malicious vendors $\{\tilde{v}\}$.

Items. Items are any goods or services sold by a vendor in a DAM. Let $\mathcal{I}$ be the set of all available items, and let $i \in \mathcal{I}$ be an item. Each vendor v may control several items. Let $\mathcal{I}_v$ be the set of item listings v controls. Let $\mathcal{I}_\tau$ be the set of

Reviews. Reviews $\mathbf{r}$ are left by customers for an item. Let $\mathcal{R}$ be the set of all reviews in the system and $\mathcal{R}_{(C,i)}$ the set of all reviews left by customers $C \subset \mathcal{C}$ for item listing i. $\mathcal{R}_{(c,\cdot)}$ denotes the set of all reviews left by c for all items, and $\mathcal{R}_{(\cdot,i)}$ denotes the set of all reviews left by all customers for item i.

Ledger. The ledger is a record of all transactions that have happened in the network, denoted $\mathcal{L}$. Our model merely assumes the existence of the ledger $\mathcal{L}$, but is agnostic with respect to how $\mathcal{L}$ is instantianted. In our implementation, we will assume $\mathcal{L}$ is a distributed record similar to Bitcoin's "blockchain;" and that $\mathcal{L}$ also includes any public parameters of the DAM.

2) Basic transactions: Customers and vendors in a DAM interact with each other via *transactions*. At minimum, a DAM needs to support (1) registration, (2) payment, and (3) review transactions, which eventually become part of the ledger $\mathcal{L}$.

Registration transaction. This transaction is used by vendors to add an item to the list of available items:

- Inputs:
  - Vendor's payment information
  - Item description
- Outputs:
  - Registration transaction **rt**

Each **rt** contains enough information to allow customers to buy the item, and the set of all registration transactions should be isomorphic to $\mathcal{I}$.

Payment transaction. This transaction moves funds from a customer to a vendor. A payment in a DAM is specified for the purchase of a particular item:

- Inputs:
  - Registration transaction **rt**
  - Payment in currency
- Outputs:
  - Payment transaction **p**

We assume that registration transaction **rt** contains all the information necessary to make a payment. Let $\mathcal{P}$ be the set of all payment transactions in the ledger, and $\mathcal{P}_{(C,i)}$ be the set of all transactions for item i by customers $C \subset \mathcal{C}$. $\mathbf{p} \in \mathcal{P}$ indicates a particular transaction, and $\mathbf{p}_c$ is a payment left by customer c.

Review transaction. A customer generates this transaction when she wants to leave a review for an item. Because reviews are exclusively generated by review transactions, we use the two terms interchangeably. Each review must be associated with a valid payment transaction.

- Inputs:
  - Payment transaction **p**
  - Review (numeric rating, textual feedback, etc.)
- Outputs:
  - Review transaction $\mathbf{r_p}$

$\mathbf{r_p} \in \mathcal{R}$ indicates a particular review associated with **p**.

3) Operations on the ledger: Any customers (both honest and adversarial) can interact with the ledger (and thus the marketplace) by using the following functions.

Add transaction $\mathbb{T}$. $\mathbb{T}$ is used to record a transaction $\mathbf{t}$ to a ledger and takes in as arguments a transaction $\mathbf{t}$ and the current state of ledger $\mathcal{L}$, and outputs a bit b indicating if the operation succeeded and a new ledger state $\mathcal{L}'$; $\mathcal{L}'$ includes $\mathbf{t}$ if $b = 1$, and $\mathcal{L}' = \mathcal{L}$ if $b = 0$.

$$\mathbb{T}(\mathbf{t}, \mathcal{L}) \to (b, \mathcal{L}').$$

Enumerate ledger $\mathbb{ET}$. $\mathbb{ET}$ is used to enumerate certain entries on the ledger. Apart from the ledger, $\mathbb{ET}$ takes in the type of transactions to enumerate and τ, a tag (which may be $\perp$) that guide the enumeration and outputs a set $\mathcal{S}$, a set of transactions of type TYPE.

$$\mathbb{ET}(\text{TYPE}, \tau, \mathcal{L}) \to \mathcal{S},$$

$\mathbb{ET}$ must support at least enumeration of types REGISTRATION and REVIEW: the first to get a list of available items for purchase, and the second to get a reputation for the item listing. For type REVIEW, τ is an item listing i for which the corresponding reviews are to be enumerated. For type REGISTRATION, τ is an associated keyword.

B. DAM Properties

A DAM must satisfy the following properties for functionality and security. We use λ to denote the security parameter. All properties are defined with respect to a probabilistic polynomial time (PPT) adversary.

P1. Correctness: Any customer c who performs a correct payment transaction $\mathbf{p}_{(c,i)}$ for an item listing i must be able to leave a review for item i. This property ensures no customer is tricked into paying a vendor without the ability to review her experience. We cannot in general promise fair trade for physical goods in a digital marketplace; we can only ensure that there is a way to report such behavior. More precisely,

$$\Pr\left[\ \mathbb{T}(\mathbf{r_p}, \mathcal{L}') \to (1, \mathcal{L}'') \ \middle|\ \mathbb{T}(\mathbf{p}, \mathcal{L}) \to (1, \mathcal{L}')\ \right] = 1\ .$$

P2. Soundness: A customer c cannot leave a review for an item i without having performed a valid payment transaction $\mathbf{p}_{(c,i)}$, and is only able to leave exactly one review per correctly formatted transaction. This ensures that customers cannot falsely lower or boost reputation of vendors beyond what their transactions entitle them to. More precisely,

$$\Pr\left[\ \mathbb{T}(\mathbf{r_p}, \mathcal{L}) \to (1, \mathcal{L}') \ \middle|\ \mathbf{p} \notin \mathcal{P}\ \right] \leq \text{negl}(\lambda)\ ,$$

and

$$\Pr\left[\ \mathbb{T}(\mathbf{r'_p}, \mathcal{L}'') \to (1, \mathcal{L}''') \ \middle|\ \begin{array}{c}\mathbb{T}(\mathbf{p}, \mathcal{L}) \to (1, \mathcal{L}') \\ \mathbb{T}(\mathbf{r_p}, \mathcal{L}') \to (1, \mathcal{L}'')\end{array}\ \right] \leq \text{negl}(\lambda).$$

P3. Item listing completeness: Any customer or vendor should be able to learn $\mathcal{I}$ efficiently. This implies that (1) it is impossible to hide any item listings from c, and that (2) it is impossible to convince c of the existence of invalid or fake item listings. This property prevents, for instance, adversaries from concealing item listings from competitors to unfairly attract customers. In other words, an item listing is complete if

$$\mathcal{I} = \mathbb{ET}(\text{REGISTRATION}, \perp, \mathcal{L}).$$

P4. Review completeness: Any customer or vendor should be able to efficiently enumerate $\mathcal{R}_{(\cdot,i)}\ \forall i$. This implies that (1) it is impossible to hide any reviews from any c and that (2) it is impossible to convince c of the existence of invalid or fake reviews. A review listing is complete if for all $i \in \mathcal{I}$,

$$\mathcal{R}_{(\cdot,i)} = \mathbb{ET}(\text{REVIEW}, i, \mathcal{L}).$$

P5. Review-payment unlinkability: This property is the most important anonymity property of a DAM. Payments (i.e., purchases) may entail sensitive information, such as the address of the customer. The adversary learning the identity of the reviewer *and* the content of the reviews, in particular negative reviews, may have bad consequences for the customer. A DAM should protect the customers by ensuring unlinkability of reviews and payments.

Let $\mathbf{P}$ be the *anonymity set* of some users' reviews; i.e., the set of payments that could have been used to generate the reviews, where $K = |\mathbf{P}|$ is the size of the anonymity set. We say that reviews and payments are unlinkable the adversary's advantage in winning the following game with a challenger is negligible in the implicit security parameter for every item. The adversary and the challenger are both given as inputs $\mathcal{L}$, $\mathbf{P}$, and any other public parameters.

1) The adversary selects the subset of honest payments $\mathbf{P}_{\text{honest}} \subset \mathbf{P}$, where $h = |\mathbf{P}_{\text{honest}}|$ and $h \geq 2$.
2) The adversary generates the set of content for reviews $M = \{m_i : i \in [h]\}$.
3) The adversary sends $\mathbf{P}_{\text{honest}}$ and M to the challenger.
4) Given the two sets, the challenger:
   a) arbitrarily assigns each $m_i \in M$ to a payment $p_j \in \mathbf{P}_{\text{honest}}$, and
   b) generates a set of review transactions $\mathbf{R}_{\text{honest}}$ given the assignment, and sends $\mathbf{R}_{\text{honest}}$ to the adversary.
5) The adversary selects any two reviews $\mathbf{r}_0$ and $\mathbf{r}_1$ from $\mathbf{R}_{\text{honest}}$, and sends them to the challenger.
6) The challenger samples a bit b. If $b = 0$, then he sends $\mathbf{p}_0$ associated with $\mathbf{r}_0$ to the adversary. Otherwise, he sends $\mathbf{p}_1$ associated with $\mathbf{r}_1$.
7) The adversary makes a guess b' for value of b.

We define the advantage to be $|\Pr[b' = b] - \frac{1}{2}|$. We allow the adversary to choose the message content, since even if the adversary knew the contents of the reviews, the relationship between payments and reviews should remain unknown.

The anonymity set $\mathbf{P}$ of particular reviews will depend on the concrete instance of DAM. The definition above guarantees honest customers' anonymity within the anonymity set. Thus, the strength of anonymity in a DAM depends directly on the size of $\mathbf{P}_{\text{honest}}$, which in practice is likely related to the size of $\mathbf{P}$. We would therefore like $\mathbf{P}$ to be equal to $\mathcal{P}_{(\cdot,i)}$ so that an honest user's review cannot be distinguished from any other honest users' reviews for an item. However, this may not necessarily be the case, and we discuss some challenges in ensuring $\mathbf{P} = \mathcal{P}_{(\cdot,i)}$ in §VI and VIII.

P6. Payment-payment (review-review) unlinkability: An adversary $\mathcal{A}$ given two payment (reviews) transactions generated by honest customers should be not able to tell if they were left by the same customer or different customers. This protects customers' identities, in case collection of payments or reviews reveal information about the customer. We say that payments or reviews are unlinkable if the adversary's

advantage in winning the following game with a challenger is negligible in the implicit security parameter for each item. The adversary and the challenger are both given as inputs $\mathcal{L}$ and any other public parameters. The challenger also knows which customer generated which payment (or review). transaction.

1) The adversary selects the subset of honest payments $\mathbf{P}_{\text{honest}} \subset \mathcal{P}$ where $h = |\mathbf{P}_{\text{honest}}|$ and $h \geq 2$, and sends the set to the challenger.
2) Given the set of transactions, the challenger informs the adversary which customers generated which transactions not in $\mathbf{P}_{\text{honest}}$.
3) The challenger figures out for each $\mathbf{p} \in \mathbf{P}_{\text{honest}}$, if there is $\mathbf{p}'$ in the set that is also generated by the same customer. We call such $\mathbf{p}'$ a *partner payment*. The challenger tells the adversary (1) if no payment has a partner payment, and (2) if *all* payments are partners of each other.
4) The challenger randomly samples a bit b, and then samples two payments $\mathbf{p}_0, \mathbf{p}_1$ in the following way.
   - If $b = 0$ and not all payments are partners of each other, then he samples a two random payments generated by different customers.
   - If $b = 0$ and all payments are partners of each other, then he samples any two payments.
   - If $b = 1$ and some payments have partner payments, then he samples a two random partner payments.
   - If $b = 1$ and there are no payments with partner payments, then he samples any two random payments.

   The challenger sends the two payments to the adversary.
5) The adversary makes a guess b' for value of b.

We define the advantage to be $|\Pr[b' = b] - \frac{1}{2}|$. We can also define a similar notion for reviews, where all the payments in the above game is replaced with a review. Note that if either conditions in Step 3 above is satisfied, then the challenger will randomly sample $\mathbf{p}_0$ and $\mathbf{p}_1$ from exactly the same set of payments, and thus the adversary can only randomly guess for b. If neither conditions are satisfied, then the challenger will sample from different sets of payments, and our goal would be to show that even here the adversary has negligible advantage.

Finally, we have a few optional properties that, while not crucial to functionality or security, benefit the customer and improve usability.

O1. Open enrollment: Anyone should be able to efficiently join or leave the marketplace as a customer or a vendor at any time. This is a requirement for a truly decentralized marketplace, as there will not be a trusted third party monitoring membership. This does not imply, for instance, that the vendor can create an item listing i at no cost.

O2. Selective review linkability: A customer c leaving a review $\mathbf{r}$ should have the option to link $\mathbf{r}$ to a set of reviews $\mathcal{PR}_c$ she has left for other items, for any $\mathcal{PR}_c \subset \mathcal{R}_{(c,\cdot)}$, in an efficient publicly verifiable way. This allows the customer to build reputation as well, by showing good reviews she has left previously; for instance, this enables the customer to potentially create several groups of reviews and convince others of her expertise in a category of products without revealing her purchases in other categories. While reviews that are publicly linked no longer satisfy P6, any other reviews must still satisfy P6.

O3. Review exculpability: Related to the previous property, a customer c leaving a review $\mathbf{r}$ should not be able to link $\mathbf{r}$ to any $\mathbf{r}' \notin \mathcal{PR}_c$. This means that a customer c should not be able to fraudulently benefit from others' good reviews by linking her review with other reviews which are not hers, or to purposefully link another customer to set of bad reviews.

IV. System Overview

Beaver uses existing blockchains and anonymous payment schemes, such as Zerocash [6], to instantiate a DAM. We present the high-level design (§IV-A), and threat model of Beaver (§IV-B). We then discuss the idea of consensus (§IV-C), and cryptographic primitives used in Beaver (§IV-D).

A. High-level design

There are three types of participants in Beaver: customers, vendors, and a distributed network of miners, all of whom enjoy open enrollment (i.e., there is no trusted third party verifying identities). The distributed network of miners and the public ledger ensure the customers' ability to leave reviews for their interactions with vendors, and allow anyone to enumerate the feedback. Beaver does not impose specialization of roles: customers and vendors could also be miners and vice-versa.

At a high level, Beaver works as follows. The vendors first register themselves to the network (i.e., the ledger) by publishing their pseudonyms. The customers are then able to enumerate the list of vendors, and purchase a product by making an anonymous transaction to the vendor. To leave a review, the customer privately ties the review to the transaction she made earlier, and submits the review to the network. Beaver, by using cryptographic primitives we will discuss later, guarantees that the clients cannot use the same transaction twice to sign a review. Finally, anyone can check the ledger to enumerate the reviews.

One key insight of Beaver is that with a public ledger, there is irrefutable public evidence that a valid transaction has taken place, and only the customer knows the secret information regarding the origin the transaction (i.e., private key used to sign a transaction). Using this, we can prevent (1) situations where a customer could be tricked into sending money but be unable to leave a review, and (2) anyone other than the real customer from leaving the review.

B. Threat model and assumptions

Beaver relies on reaching global consensus on the ledger, which contains important information such as payment transactions and reviews. We therefore require the underlying consensus protocol to be secure. In the case of a blockchain-based ledger, similar to that used by Bitcoin [36], we require at least that the adversary does not control a majority of the computational resources in the network, and that the majority of the computational resource behave rationally. Recent work on "selfish mining," however, has shown that a simple majority may not be sufficient, and one may need as much as 75% of the network to be honest [23]. In any case, our security assumption will be identical to the security assumption for the underlying public ledger scheme. Beaver also assumes that the customers and vendors are rational, and do not behave maliciously if the cost of doing so is significant. Apart from these two assumptions, we do not limit the adversary's power. The adversary could, for instance, control many vendors and customers that collude with each other, and try to boost her own ratings or lower competitors' reputations.

We also assume existence of an anonymous payment system, which allows anonymous transactions and anonymous transfers of coins back and forth from a regular cryptocurrency like Bitcoin. Zerocash [6], for instance, is a candidate for such currency. Finally, we assume that any communication, especially that of the customers, is done via a truly anonymous communication to ensure anonymity. In practice, the customers may use Tor [20] or other stronger anonymous communication systems [30], [48]. These systems may not behave as an ideal anonymous communication in reality, but addressing this issue is outside the scope of this paper.

C. Consensus

Beaver makes use of a consensus protocol for establishing network-wide agreement about the state of the marketplace such as item listings, reviews, and any other actions of its members. In particular, we make extensive use of a public ledger like that of Bitcoin [36], which uses proofs of work to arrive at a global consensus. While there are several other consensus protocols in the literature, the public ledgers in Bitcoin make relatively small assumptions about the network, making it a prime candidate for a decentralized application like Beaver.

At the core of Bitcoin-style public ledger is a hash chain that is constructed by a distributed set of miners. For a period of time, miners listen to messages being broadcast in the network by users, such as a transaction to transfer money from one account to another. Miners then aggregate these messages into a block along with the hash of the previous block and enter it into the public ledger by performing a proof-of-work. In Bitcoin, this proof-of-work is finding a nonce, such that the hash $H(\text{BLOCK}||\text{NONCE}) < \alpha$ where α is a parameter that determines the difficulty of the proof-of-work.

Miners in such systems are assumed to be economically rational actors, and so they need to be incentivized to spend their computational resources on mining blocks for the network. To do this, Bitcoin has (1) a reward for mining a block, and (2) a transaction fee. Beaver will rely on similar incentives to encourage miners to behave honestly and maintain a healthy ledger.

D. Cryptographic primitives

Beaver employs two cryptographic primitives, non-interactive zero-knowledge proofs and linkable ring signatures, which we describe next.

1) NIZK: A non-interactive zero-knowledge proof, or a NIZK, of a statement is a zero-knowledge proof of the statement that could be verified easily by anyone without interaction with the party who generated the proof. In Beaver, we use a subcategory of NIZK called non-interactive zero-knowledge proof-of-knowledge (NIZKPoK), used to prove knowledge of a secret value. NIZKPoKs are commonly used to show that, given a blinded version of a secret value (e.g., a commitment), whoever generated the proof knows the secret value underlying the commitment. An example of NIZKPoK is a zero-knowledge proof of possession of discrete logs: given g^x for a generator g of a group $\mathbb{G}$ in which discrete log is hard, NIZKPoK can be used to prove knowledge of x without revealing any information on x. This can be done by applying the Fiat-Shamir heuristic [24] to a standard zero-knowledge proof of discrete log [13].

2) Linkable ring signatures: Ring signatures (RSig), first proposed by Rivest *et al.* [39], are cryptographic signatures that guarantee anonymity of the signers. Specifically, a RSig algorithm takes as input the private key of the signer, a set of public keys, and a message, and generates a signature that can be verified against the *set* of public keys without revealing *which* key was used to sign the message.

Ring signatures unfortunately do not offer any form of accountability, and there is no way to stop the signer from signing multiple times even when it is not desirable. Linkable ring signatures (LRSig, [31]), on the other hand, are accountable variants of ring signatures: All signatures generated by the same signer can be linked to each other, though the identity of the signer is still hidden. LRSigs can be used to prevent signers from signing multiple times, while preserving anonymity.

V. System Design

In this section, we describe the available operations in Beaver and the typical workflow which is shown in Figure 1. We define registration transactions (§V-A), special vendor transactions (§V-B), payment transactions, (§V-C), and review transactions (§V-D). In each section, we present the details of the transactions and how the miners could verify them. All transactions are signed with an unforgeable signature scheme by the party generating the transaction, unless otherwise noted.

A. Registration

Similar to Bitcoin [36] and other cryptocurrencies, Beaver does not require explicit registration for miners. This is also true for customers in Beaver, who by our design choices and the asymmetry of DAMs, do not need any publicity. In some usage cases of a DAM, for example in the context of Uber, where a driver (vendor) makes physical contact with a customer (passenger), the vendor may wish to query the reputation of passenger before agreeing to the transaction and is discussed in §VIII.

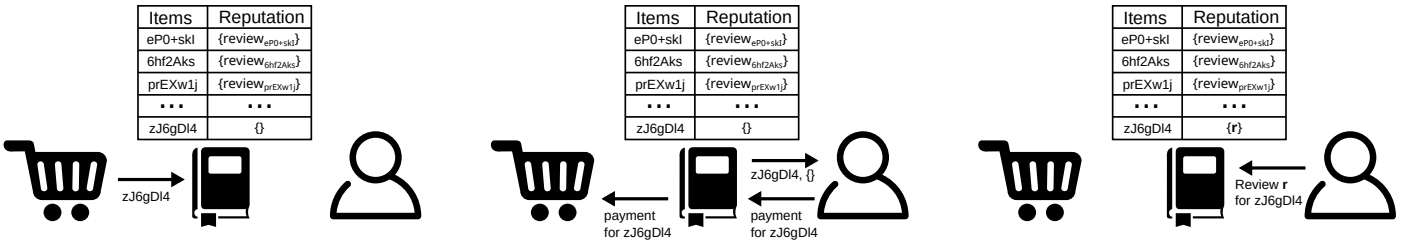
A vendor, on the other hand, needs a public identity (pseudonym) that others could refer to for purchases and reviews. We make the following design choice in Beaver. Instead of explicitly defining a concept of "vendor reputation" *per se*, the reputation of a vendor v is bound to the reputation (reviews) acquired for all the items in $\mathcal{I}_v$ the vendor is selling. So, *items* end up being the primary reputation vector—as we will see later, vendors can link various items they sell which each other to create a vendor profile.

A vendor registers an item i that she wants to sell by generating a new, *unique*, public-private key pair (i_{pk}, i_{sk}) in the underlying payment system that will be used to receive payments for this item. The vendor then covers an item registration fee f_{reg} by moving money into i_{pk} and forms a registration transaction

$$\mathbf{rt} = (\text{REGISTRATION}, \text{TXID}, \text{ITEMINFO}, i_{pk})\,.$$

REGISTRATION is the type of transaction, and TXID is a unique identifier for the transaction which in practice could the hash of all the other values in **rt** (including the unique public key i_{pk}). ITEMINFO is the description of the item being sold, the price, and any other information needed to generate a payment, and i_{pk} is the public key uniquely associated with this item.

Once the miners receive this transaction, they run Algorithm 1 to verify the transaction before adding it to the



(a) Vendor registers the public key of item to the ledger.

(b) Customer retrieves item listings and their reviews, and makes a payment transaction for an item.

(c) Customer leaves a review for the item purchased (optional).

Fig. 1: Beaver workflow

ledger. The miner that successfully adds **rt** to $\mathcal{L}$ claims the fee f_{reg} from i_{pk}. Once added, any customer can find the list of all items sold in Beaver by enumerating all REGISTRATION transactions. The customer can then purchase the product by sending money to the public key in the registration transaction. Moreover, the customers can check all available reviews for this key (item) as detailed in §V-D.

Algorithm 1 Registration verification

INPUTS:
1) **rt** $= (\text{REGISTRATION}, \text{TXID}, \text{ITEMINFO}, i_{pk})$
2) $\mathcal{L}$

OUTPUT: Miners add **rt** to $\mathcal{L}$ only if all of the steps are satisfied.
1) Check i_{pk} has enough funds to cover the registration fee f_{reg}.
2) Check $\text{TXID} \notin \mathcal{L} \wedge i_{pk} \notin \mathcal{L}$.
3) Check ITEMINFO specifies the price of the item x, and that the price is within minimum and maximum denominations of underlying currency.

B. Special vendor transactions

Vendors in Beaver may also perform two special transactions: (1) bootstrapping reputation and (2) updating a listing.

Bootstrapping reputation.

When an item is first listed, it has no reviews. The vendor, however, may have other products with positive reviews and may want to bootstrap the reputation for this new listing by linking it to its other items. While these reviews not directly express the quality of the particular item, they could help customers establish trust.

To perform this linkage, the vendor can submit a special transaction that includes a NIZKPoK of the private keys of other items she sells. The customers can then check the reviews of the other items, and be convinced that these items come from the same vendor. Note that this transaction may be submitted at any time, and the vendor may choose when to link the items together.

$$\textbf{br} = \left(\text{BOOTSTRAP}, \text{TXID}, i_{pk}, (\overrightarrow{j_{pk}, \pi_{j_{sk}}})\right)$$

i_{pk} is the public key associated with the item listing that the vendor is interested in bootstrapping and $(\overrightarrow{j_{pk}, \pi_{j_{sk}}})$ is a vector of public keys associated with other item listings $j \in \mathcal{I}_v$, and NIZKPoK for the secret keys to each. Miners use Algorithm 2 to verify this transaction. The miner that successfully adds **br** to $\mathcal{L}$ claims a bootstapping fee f_{boot} from i_{pk}.

The NIZKPoKs, if generated carelessly, could be susceptible to replay attacks. That is, a malicious vendor could reuse the NIZKPoKs of honest vendors to falsely bootstrap their reputation. To prevent this, we suggest using a public nonce when generating the NIZKPoKs. Concretely, when one applies Fiat-Shamir heuristic to generate a NIZKPoK, the input to the random oracle (i.e., the cryptographic hash function) should be a public nonce along with the standard input for the heuristic. The nonce need not be random, but needs to be publicly computable, non-repeating, and not under the control of the adversary to prevent replay attacks. For example, one could use the TXID as the nonce. For the rest of this paper, we assume that this is done for all NIZKPoKs generated.

Algorithm 2 Item listing bootstrap verification

INPUTS:
1) **br** $= \left(\text{BOOTSTRAP}, \text{TXID}, i_{pk}, (\overrightarrow{j_{pk}, \pi_{j_{sk}}})\right)$
2) $\mathcal{L}$

OUTPUT: Miners add **br** to $\mathcal{L}$ only if all steps are satisfied.
1) Check i_{pk} has enough funds to cover the fee f_{boot}.
2) Check $\text{TXID} \notin \mathcal{L}$.
3) From $\mathcal{L}$, verify that i_{pk} and all the j_{pk}'s are valid item public keys.
4) For each component $(j_{pk}, \pi_{j_{sk}})$ of $(\overrightarrow{j_{pk}, \pi_{j_{sk}}})$, verify $\pi_{j_{sk}}$ is a valid NIZKPoK for the secret key j_{sk}.

Updating listings. A vendor may want to update an item listing after creating it. For example, the item may be sold out, discontinued, need a description change, or the vendor may wish to hold a promotion or sale. The vendor can issue an update by submitting a special transaction:

$$\textbf{ut} = (\text{UPDATE}, \text{TXID}, i_{pk}, \pi_{i_{sk}}, \text{ITEMINFO})$$

Here i_{pk} is the public key of the item listing that a vendor wishes to update, $\pi_{i_{sk}}$ is a NIZKPoK for the corresponding secret key, and ITEMINFO is the new information for the item listing. Miners use Algorithm 3 to verify this transaction, and the miner that successfully adds **ut** to $\mathcal{L}$ claims an update fee f_{upd} from i_{pk}.

Algorithm 3 Item listing update verification

INPUTS:
1) **ut** $= (\text{UPDATE}, \text{TXID}, i_{pk}, \pi_{i_{sk}}, \text{ITEMINFO})$
2) $\mathcal{L}$

OUTPUT: Miners add **ut** to $\mathcal{L}$ only if all steps are satisfied.
1) Check i_{pk} has enough funds to cover the fee f_{upd}.
2) Check $\text{TXID} \notin \mathcal{L}$.
3) From $\mathcal{L}$, verify that i_{pk} is a valid public key for an item.
4) Verify that $\pi_{i_{sk}}$ is a valid NIZKPoK of the secret key for i_{pk}.

C. Payments

When a user c decides to purchase an item, she begins by generating a fresh public key/private key pair (c_{pk}, c_{sk}). As in Bitcoin, in which public "wallet" addresses are public keys, c_{pk} is basically equivalent to a pseudonym; while we use the notation c_{pk} here for convenience, a given customer c will actually hold many such public-private key pairs—essentially one per transaction. The user then transfers funds anonymously (e.g., via Zerocash [6]) to c_{pk}. She finally transfers funds from c_{pk} to the public key associated with the item she wishes to purchase. She may use the transaction to supply other relevant information such as a shipping address or any special requests for the order, or she may send that information out of band (e.g., through a vendor's website), along with the proof of the purchase via anonymous communication.

In Beaver, we do *not* provide transaction privacy for the vendor; i.e., the recipient of the payment transaction is not hidden, unlike in Zerocash. This is done so that the users can understand the explicit anonymity set when leaving a review. The implications of this design choice are discussed in §VIII.

A payment transaction **p** looks similar to that of Bitcoin:

$$\mathbf{p} = (\text{PAYMENT}, \text{TXID}, c_{pk}, i_{pk}, \hat{x}, \text{CUSTOMERINFO})$$

where PAYMENT is the type of transaction, TXID is the unique transaction ID, c_{pk} is the customer's fresh pseudonym (public key) that holds enough funds to pay amount $\hat{x}$ to the vendor's account i_{pk} as well as any additional fees. $\hat{x}$ is *at least* the price x of the item i_{pk} but could be more, for instance if several quantities of the items are purchased, or if the customer wishes to include a tip for the vendor. If the underlying cryptocurrency supports adding supplementary information to transactions, then CUSTOMERINFO is passed along as information specific to the order such as a shipping address or special requests encrypted under i_{pk}. One key difference between payment transactions in Beaver and payment transactions in Bitcoin is that the transaction fee of the payment is broken into two fees: a *tax* f_{tax} and a *reviewing fee* f_{rev}. f_{tax} is paid to the miner who adds the payment transaction to the blockchain, similar to transactions fees in traditional cryptocurrencies. f_{rev} is paid later to the miner who adds the review associated with this payment to the blockchain (§V-D). Upon submission, miners use Algorithm 4 to verify that the payment satisfies all requirements, and the miner that successfully adds **p** to $\mathcal{L}$ claims f_{tax} from c_{pk}. Any future transfer from c_{pk} to another place will be considered invalid, except to claim the reviewing fee, for reasons we describe next.

Algorithm 4 Payment verification

INPUTS:
1) **p** $= (\text{PAYMENT}, \text{TXID}, c_{pk}, i_{pk}, \hat{x}, \text{CUSTOMERINFO})$
2) $\mathcal{L}$

OUTPUT: Miners add **p** to $\mathcal{L}$ only if all of the steps are satisfied.
1) Check that the available funds in c_{pk} is larger than $\hat{x} + f_{\text{tax}} + f_{\text{rev}}$.
2) Check $\text{TXID} \notin \mathcal{L}$.
3) From $\mathcal{L}$, verify that i_{pk} is a valid public key for an item.
4) Find ITEMINFO for i_{pk} in $\mathcal{L}$, and the price x for i_{pk}.
5) Check $\hat{x} \geq x$.

D. Reviews

After the payment transaction **p** is added to the blockchain, the customer has the option to form a review for the item. The review will contain a message from the customer (e.g., a detailed product review) as well as a numeric rating a, expressed as a small integer. Once she writes the review, the customer can then sign the review with the private key c_{sk} used to make the payment, and send it to the miners.

A naïve signature would reveal the transaction associated with the private signing key c_{sk}, and would therefore tie the transaction to the review. Unfortunately, the transaction may contain sensitive information about the customer, which may limit the customer's ability to leave truthful feedback. Though we wish to hide the exact relationship between reviews and transactions, we must also ensure that there exists a valid payment associated with each review. To achieve this objective, we use linkable ring signatures. The list of all payments left for i_{pk} is first publicly divided into groups of size K, where K is a public system parameter. For instance, the first K payments form the first group, the second K payments form the next group, and so on. The customer c, using c_{pk}, who wishes to leave a review for the item i_{pk} figures out the group that her payment transaction **p** belongs to, takes all public keys of customers within that group, and uses them to sign the review with an LRSig instead of a regular signature. With an LRSig, no one is able to learn which one of the K payments is linked to the review. An LRSig also guarantees that any attempt to submit more than one review per payment will be caught via the linkability of signatures.

With this change, we have to modify the reviewing fee slightly, as the miner who adds the review to the chain cannot figure out which transaction should pay the reviewing fee. Instead, the miners can collect this reviewing fee from any transaction in the same group that has not yet been claimed, as all valid transactions have been specified to pay the reviewing fee. Furthermore, as before, the miners will reject any attempt to transfer the reviewing fee to another account before the fee is legitimately claimed by a miner, ensuring there is enough funds in the account to pay the miner. Although there could be a delay in waiting for K transactions to appear on the chain, once there are sufficiently many transactions, the reviews could be submitted at any time (i.e., asynchronously) and still guarantee the anonymity among the set of K transactions.

Customer reputation. In existing e-commerce systems like Amazon and eBay, it is possible for a customer to build up her reputation as a "good" reviewer or an "expert" of a

category of products, by linking all the reviews she leaves to her account. For instance, if one person has left reviews for many different headphones and speakers, then that person's reviews for another set of headphones may be more valuable to prospective customers. However, a customer may also desire the property that reviews for certain products do not link back to other products she has reviewed for sake of privacy.

In Beaver, we allow the customer to choose which reviews to tie together: When leaving a review, the customer also generates a random value r_c that she keeps secret and its commitment $\texttt{Comm}(r_c)$, that she includes in the review. Unlike with special vendor transactions, a customer cannot use NIZKPoKs for her secret key since that would require revealing the public key. Instead, a customer who wishes to link reviews together may refer to a previous review containing $\texttt{Comm}(r_c)$ and include the NIZKPoK of r_c in the new review.

Review revocation. The reviewers may want to update their reviews of a product after some time. Often, for example, a product works well in the first few days, and the customer leaves a positive review. Soon after, the product breaks, and the customer may desire to change the review to something negative. In these scenarios, the reviewer simply re-signs the review, using Algorithm 5 with the same private key and public keys, and sends the new review to the miners. When enumerating the reviews, if anyone finds reviews that are linked (via LRSig), then she simply takes the latest review and ignores older ones. This review, however, must be submitted with a separate reviewing fee to incentivize the additional work needed from the miners.

In summary, a review transaction $\mathbf{r}$ is the tuple

$$\mathbf{r} = \left(\text{REVIEW}, \text{TXID}, a, M, c'_{pk}, i_{pk}, \texttt{Comm}(r_c), \overrightarrow{c_{pk}}, \vec{z_\ell}, \vec{\mathbf{r}_\ell}, \sigma\right)$$

where REVIEW and TXID are the type and ID of the transaction, a is the numerical rating, M is the detailed review, and c'_{pk} is a fresh pseudonym to draw the review fee from if this review is an update. If this review is not an update, c'_{pk} and $\sigma_{c'_{sk}}$ may both be $\texttt{null}$. i_{pk} is the public key of the item of the review, $\texttt{Comm}(r_c)$ is the commitment of a random value used to link reviews if desired later on, $\overrightarrow{c_{pk}}$ the set (of size K) of public keys, that includes the customer's own public key c_{pk} (along with others). $\vec{\mathbf{r}_\ell}$ contains the other reviews the customer c has previously written and wishes to link to this new review, if any; and $\vec{z_\ell}$ contains the corresponding NIZKPoKs of the secret random values in the reviews in $\vec{\mathbf{r}_\ell}$. $\vec{z_\ell}$ and $\vec{\mathbf{r}_\ell}$ may be empty, if the customer does not wish to link any reviews. Finally, σ is a signature on all other values in $\mathbf{r}$ with the secret key corresponding to c'_{pk}. Customers use Algorithm 5 to generate reviews, and the miners run Algorithm 6 to verify the reviews before adding them to the ledger.

Algorithm 5 Review generation

INPUTS:
1) (c_{pk}, c_{sk}): public-private key of the customer
2) c'_{pk}: public key of pseudonym to pay reviewing fee (update only)
3) i_{pk}: public key for the item listing
4) a: rating for this item
5) M: short message for this review
6) $\vec{\mathbf{r}_\ell}$: set of reviews to link to
7) $\vec{r_\ell}$: secret random values in reviews to be linked
8) $\mathcal{L}$: public ledger

OUTPUT: A review transaction and a linkable ring signature σ of the review for item i_{pk}.
1) From $\mathcal{L}$, verify that i_{pk} is a valid public key for an item.
2) Divide payment transactions for $i_{pk} \in \mathcal{L}$, into (public decided) groups of K.
3) Find the group that c_{pk} belongs to, and extract the K public keys to yield $\overrightarrow{c_{pk}} = \overrightarrow{(c_{j,pk})_{j\in[K]}}$.
4) Generate a random value r_c and its commitment $\texttt{Comm}(r_c)$.
5) Let $\overrightarrow{\texttt{Comm}(r_\ell)}$ be the commitments to random values in other reviews to be linked. Note that if the customer does not wish to link reviews, then $\vec{z_\ell} = \texttt{null}$, $\vec{\mathbf{r}_\ell} = \texttt{null}$.
6) Generate review $\mathbf{r}' = \left(\text{REVIEW}, \text{TXID}, a, M, c'_{pk}, i_{pk}, \texttt{Comm}(r_c), \overrightarrow{c_{pk}}, \vec{z_\ell}, \vec{\mathbf{r}_\ell}\right)$.
7) Compute signature $\sigma = LRSig\left(\mathbf{r}', c_{sk}, \overrightarrow{c_{pk}}\right)$.
8) Output $\mathbf{r} = (\mathbf{r}', \sigma)$.

Algorithm 6 Review verification

INPUTS:
1) $\mathbf{r} = \left(\text{REVIEW}, \text{TXID}, a, M, c'_{pk}, i_{pk}, \texttt{Comm}(r_c), \overrightarrow{c_{pk}}, \vec{z_\ell}, \vec{\mathbf{r}_\ell}, \sigma\right)$
2) $\mathcal{L}$

OUTPUT: Miners add $\mathbf{r}$ to $\mathcal{L}$ only if all steps are satisfied.
1) Check $\text{TXID} \notin \mathcal{L}$.
2) From $\mathcal{L}$, verify $\overrightarrow{c_{pk}}$ is a valid set of public keys associated with K transactions to i_{pk}.
3) Verify σ on $\mathbf{r}$.
4) Verify c'_{pk} has at least f_{rev} funds to cover the review fee.
5) If $\vec{z_\ell}$ is not $\texttt{null}$, then verify the NIZK.

E. Transaction enumeration

Beaver users will routinely want to enumerate certain types of transactions (e.g., REGISTRATIONS, REVIEWS). Enumeration operations should thus be efficient. We outline here a mechanism for users to efficiently query an untrusted node and verify its response, instead of downloading $\mathcal{L}$ themselves.

When a transaction $\mathbf{t}$ is added to $\mathcal{L}$, the miners compute the total number of transactions of that type thus far, $|\mathbb{ET}(\text{TYPE}(\mathbf{t}), \tau, \mathcal{L})|$, and append it to the transaction. For example, if a review $\mathbf{r}$ for item i is to be added to a block, miners compute $|\mathbb{ET}(\text{REVIEW}, i, \mathcal{L})|$, and append it to $\mathbf{r}$. Likewise, optional keywords in item descriptions can be used to keep track of the number of items in $\mathcal{L}$ featuring each keyword.

We then borrow ideas from Simplified Payment Verification (SPV) in Bitcoin [37]. Transactions in every block are arranged into a Merkle tree, and the root of the tree is added to the block header. This enables a user with the root to efficiently verify membership of a transaction in the block when given the Merkle proof. In addition to quick verification of membership, we may also want the users to be able to quickly check if a transaction of particular type is present in the block. To this end, we require the miners to also include in the header, Bloom filters [9] indicating (1) for every $i \in \mathcal{I}$, whether there is a review in this block for i, and (2) for every available item description keyword, whether an item registered in this block includes that keyword. Users locally only store the Merkle tree root and Bloom filters for each block, i.e., about 1–3KB of data

per block.

Untrusted nodes maintain a separate representation of $\mathcal{L}$ which is optimized for efficient queries, such as the Patricia Trees employed by Ethereum [49]. When a user wants to enumerate transactions, she queries an untrusted node, who replies with $(\mathcal{S}, \vec{\pi})$, where $\mathcal{S}$ is the result of enumeration. For each transaction in $\mathcal{S}$, $\vec{\pi}$ contains a Merkle proof that the transaction is in $\mathcal{L}$, which users can quickly verify.

While the SPV provides a way for users to be convinced that all transactions returned are indeed part of $\mathcal{L}$, a malicious node may still omit some transactions (e.g., leaving out negative reviews). To resolve this, the user can manually query the most recent transaction of the keyword τ. She first searches for the latest occurrence of the transaction associated with τ by using the Bloom filters. When a match is found, she downloads the full block and checks it for the relevant transaction, extracting the counter for the total number of transactions of the type if found. In the event that the transaction was not found in the matching block (i.e., false positive in the filter), she keeps iterating until either the transaction is found or the beginning of $\mathcal{L}$ is reached. Once she has the counter, the user can verify that $|\mathcal{S}|$ is equal to the counter in $\mathcal{L}$ to check if any transactions were omitted. A detailed evaluation of enumeration performance can be found in §VII.

VI. Analysis

We first provide proof sketches and argue Beaver satisfies all the properties of DAM (§III-B). We then analyze the impact of different parameters. In our analysis, we assume that a consensus on the $\mathcal{L}$ has been reached, and the state of $\mathcal{L}$ is publicly visible to everyone in the system. Note that the operations on the ledger $\mathbb{T}$ and $\mathbb{ET}$ are done by interacting with the miners, and is made available by underlying consensus protocol in Beaver.

A. Properties of Beaver

P1. Correctness: A payment is completed when the payment transaction $\mathbf{p}$ ends up in the ledger $\mathcal{L}$. Upon K completed transactions for an item, customers may generate a review using Algorithm 5, and the signatures will verify as long as a valid payment is in $\mathcal{L}$. Thus, assuming the underlying consensus protocol is secure, the miners will add the review to the ledger.

P2. Soundness: Soundness is derived directly from the forgery resistance and linkability of linkable ring signatures. Without a payment transaction in $\mathcal{L}$ an adversary would not have a secret key for any of the public keys associated with the K transactions. The forgery resistance of linkable ring signatures implies that this adversary cannot generate a valid signature that will verify. Similarly, if a customer signs two reviews using the same secret key, then the two signatures will be linked (via linkable property of the signatures), and anyone can detect this misbehavior.

P3. Item listing completeness: Assuming consensus and availability of $\mathcal{L}$, any item listing is publicly visible to everyone since $\mathcal{L}$ holds item listings.

P4. Review completeness: Similar to item listing, assuming consensus and availability of $\mathcal{L}$, any review is publicly visible.

P5. Review-payment unlinkability: The anonymity property of linkable ring signatures guarantees that any honest signers identity remains hidden among the other honest signers within the same anonymity set. In particular, the adversary, given two signatures (reviews) and two public keys (payments), it cannot learn which signature was signed by a private key associated with which public key better than guessing, even if it knew the messages associated with the signatures. Moreover, the optional NIZKPoKs used to link reviews only link reviews together without revealing any information about the payments or the public keys used. Therefore, the advantage of the adversary winning the game is negligible.

P6. Payment (review) unlinkability: For every purchase, customers generate a fresh pseudonym which is not linked to any other pseudonyms previously used (via security of the underlying payment scheme), therefore no purchases can be linked to other purchases. Reviews are also signed with the fresh pseudonyms, so reviews cannot be linked to each other, unless explicitly linked via NIZKPoKs.

O1. Open enrollment: As long as the underlying payment system and consensus protocol allows for open enrollment (e.g., Bitcoin), customers, vendors, and miners can all join freely.

O2. Selective review linkability: A customer who previously generated a review should know the random value in the commitment used for that review. The customer can then use NIZKPoK to prove possession of the random value in a new review to link two reviews together.

O3. Review exculpability: If the commitment scheme is hiding, then the adversary cannot learn the secret value from the commitment. Since only the honest customer knows the secret value, no other person can generate a correct NIZKPoK to link the reviews.

B. Parameters

Many parameters impact the performance and security of Beaver.

Registration fee f_{reg}. f_{reg} is the cost that a vendor must pay to register a new product. This cost mitigates the problem of vendors with negative reviews simply creating new listings to reset the reputation associated with an item. It also mitigates attacks where an adversary attempts to flood the network with item listings to make enumeration and identification of legitimate item listings tedious. However, f_{reg} should not be not set too high as to discourage new vendors from joining the network.

Transaction tax f_{tax}. f_{tax} is paid to miners when a customer purchases an item, and directly impacts the credibility of reviews in our system. Since f_{tax} is paid to the network of miners and is lost to both the customer and the vendor, high values of f_{tax} would deter adversaries from leaving fake reviews to influence someone's reputation, and would mitigate Sybil attacks. For example, f_{tax} discourages a vendor from acting as a customer and purchasing her own product to leave positive reviews. However, if f_{tax} is too high, customers may be discouraged from using Beaver and instead seek out cheaper alternatives with less overhead. We believe that f_{tax} should be proportional to the value of the transaction being made—perhaps around 10% of the item price similar to vendor fees charged by Amazon Marketplace. This insight forms the basis for formally defining the *credibility* of a review as the lower bound on the cost to an adversary of creating for creating it.

Finally, f_{tax} also incentivizes the miners to add the payment transaction to $\mathcal{L}$.

Reviewing fee f_{rev}. f_{rev} is paid to the miners for leaving a review, and also helps establish the lower bound on the cost of leaving a review in conjunction with f_{tax}. In Beaver, reviews transactions are separate from payment transactions although practically, customer need to supply funds for them both at the same time. We recommend either an absolute fixed fee for f_{rev}, or a fee proportional to the size of the review (i.e., length of the message), similar to transaction fees in Bitcoin [36].

Anonymity set size K. K is the number of transactions that must occur for an item listing before customers are allowed to leave reviews, and each review will be anonymous within the K transactions. Unforuntately, not all K transactions necessarily come from honest users, and γ of the K transactions may in fact be generated by the adversary. In these settings, we should try to estimate γ, and set K such that $K - \gamma$ is sufficiently large with respect to the perceived adversary. For example, we can use estimation of the adversary's total budget along with f_{tax} to estimate γ. We can then set K appropriately so that identifying a transaction out of $K - \gamma$ requires the adversary to spend more resources than they are willing to spend. In general, larger values of K provide stronger anonymity, at the expense of a higher review latency (see §VII).

Reputation bootstrap fee f_{boot}. Since vendors can only link the items that it knows the secret keys for (i.e., owned by the vendor), bootstrapping reputation should not be a costly operation. f_{boot} should be set to be the minimal value that would incentivize the miners to add this transaction to the $\mathcal{L}$.

Item update fee f_{upd}. A vendor should be able to update their items, so f_{upd} should be set to be the minimal value that would incentivize the miners sufficiently to add this transaction to $\mathcal{L}$. A vendor may change the price very frequently in an attempt to differentiate prices for different customers, but this is visible to the public, and the price at a given time is the same for every customer.

1) Review credibility: Customers who are interested in purchasing an item will first want to look at the reviews left by previous customers. In addition to observing the score and message of a previous review, a customer may be interested in evaluating a review's credibility, or what it would have costed an adversary to create the review himself.

While it might be tempting to assume that cost to an adversary for generating a review is the price of the item plus the associated tax f_{tax} and reviewing fee f_{rev}, it may be the case that an adversary is purchasing the item from himself, and is therefore recovering the price of the item. The cost to an adversary for generating a review is therefore lower bounded by $f_{\text{tax}} + (\ell + 1) f_{\text{rev}}$, where f_{tax} is the tax paid on the payment transaction, ℓ is the number of times the review has been updated. f_{rev} is recommended to be set to a constant value, and f_{tax} will likely scale with the price of an item. With the ability to change the price of an item (§V-B), f_{tax} may be different for different transactions, and users cannot link a review to any particular transaction. However, customers do know the lowest possible tax for a review since the K transactions that form the anonymity set are public. Therefore, customers should be conservative, and assume the lowest f_{tax} out of the K transactions to estimate the credibility of a review.

The baseline credibility, $f_{\text{tax}} + (\ell + 1) f_{\text{rev}}$, may however be augmented by other reviews that were linked. A naïve augmentation of the credibility would be to add up the credibility of all linked reviews. However, this enables an adversary to pay (out-of-band) a good reviewer to generate a NIZKPoK, and all of sudden get a review of very high credibility for potentially much less cost than estimated. There may be other out-of-band attacks that Beaver may not protect against, so customers should be careful when augmenting credibility of a review using linked reviews.

TABLE I: Size and space benchmarks for basic building block operations in Beaver

	Time	**Space**
NIZKPoK (Generate)	6.89ms	96 B
NIZKPoK (Verify)	11.91ms	-
Ring Signature, K = 1 (Generate)	13.5ms	64 B
Ring Signature, K = 2 (Generate)	21.5ms	96 B
Ring Signature, K = 10 (Generate)	133.7ms	352 B
Ring Signature, K = 100 (Generate)	1420ms	3232 B
Ring Signature, K = 1 (Verify)	13.5ms	-
Ring Signature, K = 10 (Verify)	142ms	-
Ring Signature, K = 100 (Verify)	1390ms	-
Digital Signature (Generate)	6.82ms	64 B
Digital Signature (Verify)	11.88ms	-
Moving funds to fresh pseudonym [6]	$\geq$ 2min	-

VII. Evaluation

We next evaluate Beaver's practicality in terms of computational performance and storage overhead; we also evaluate Beaver's usefulness in providing anonymity in a marketplace setting.

A. Computation and storage overhead

1) Blockchain query latency: To estimate the time to query transactions in Beaver, we first obtained (from the authors) the online anonymous (centralized) marketplace transaction dataset Soska and Christin collected [41]. This dataset represents a large ecosystem of real-world transactions over two years. We converted this entire transaction data into a blockchain, which we used to construct an indexed database optimized for fast lookup operations, something that we would encourage users (i.e., Beaver clients) to do in practice.

Queries for random transactions take less than 1 ms on the 10 GB dataset on a RAID 5 consisting of 8 solid state drives. We then upsampled the database to a size of 4 TB, and random lookups still took less than 1 ms each. This result gives us confidence that the lookup overhead will be negligible compared to other operations in the system.

2) Algorithm runtime and space: We next evaluate the performance of the basic cryptographic operations used in Beaver. For NIZKPoKs, we implemented zero-knowledge proofs for discrete logs [13] with Fiat-Shamir heuristic [24] in Go programming language using Curve25519. For LRSigs, we used the implementation of ring signatures found in the DeDiS crypto package [1].

Table I shows the time to perform each operation as well as the amount of state created. The basic operations are quite reasonable, taking just a few milliseconds of processing time on a system running an i7-3612QM @ 2.1 GHZ with 8 GB

TABLE II: Size and space benchmarks for operations in Beaver

	Time (Expected)	Space
Item Listing Transaction	7 ms	1120 B
Item Listing Verification	12 ms	-
Vendor Bootstrap Transaction	14 ms	224 B
Vendor Bootstrap Verification	12 ms	-
Listing Update Transaction	14 ms	1216 B
Listing Update Verification	12 ms	-
Payment Transaction	7 ms	1160 B
Payment Verification	12 ms	-
Review Transaction	7 + 14·K ms	1412 B
Review Verification	14·K ms	-

of RAM. The generation and verification time as well as the size of the ring signatures all scale linearly with the size of the anonymity set K. Our choice of anonymous payment scheme determines latency of the purchase. One potential candidate, Zerocash [6], yields operations more than two orders of magnitude slower than other operations in Beaver, but could be easily replaced with a faster anonymous payment scheme in the future. Moreover, customers could also move funds in advance to decrease the overall latency.

Next, we look at the performance of the higher level operations in Beaver, shown in Table II. For these operations, we fix the size of ITEMINFO, CUSTOMERINFO and M the review message to 1024 bytes. We also assume that only one proof is linked at a time and that users are running commodity hardware with 4 cores, as in practice, basic operations may be computed in parallel such as verifying the signature of a transaction and the correctness of a proof inside.

All transactions are generated and verified reasonably quickly with reviews scaling linearly with the anonymity set size due to the ring signature. All operations can be performed in constant size, but the space complexity of traceable ring signatures is typically linear in the group size. We avoid this issue by instead looking up the public keys of the anonymity set from $\mathcal{L}$ during verification, which yields ring signatures that only add constant size without the use of accumulators [42].

B. Anonymity Set Size

Crucial to the practicality of Beaver is the size of the anonymity set to which customers belong, and the latency between the times a customer buys an item and when she is able to leave a review. To evaluate these components we again use Soska and Christin's marketplace dataset to simulate Beaver in a real-world setting.

Figure 2 shows a cumulative distribution function of the latency in days from when a customer purchases an item to when they are able to leave a review for different choices of the system-wide anonymity set size parameter K. For small anonymity sets such as $K = 5$, about 80% of customers are able to leave a review within one week, while larger anonymity sets ($K = 50$ or $K = 100$) take more than 75 days when dealing with unpopular items. Despite high latency for these reviews to appear, vendors can still ensure consumer confidence by bootstrapping reputation from other item listings that they control. Similar to the notion of cover traffic in anonymous communication networks, widespread adoption of

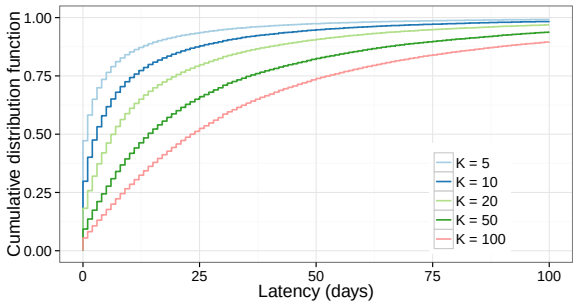


Fig. 2: Latency to leave feedback as a function of K.

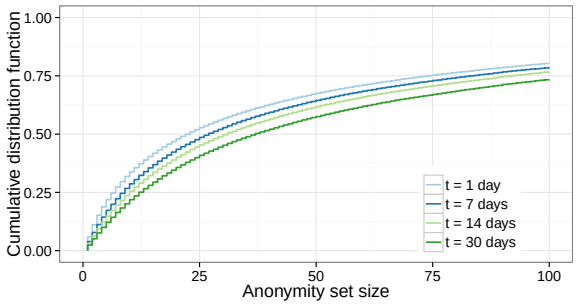


Fig. 3: Anonymity set size given minimum review latency.

Beaver and high amounts of transactions would build the anonymity sets faster, and therefore reduce latency.

We also explore adjusting K independently for each item so that it meets certain review latency requirements. Figure 3 presents cumulative distribution functions of K when we fix the time it takes for a customer to build her anonymity set (i.e., minimum review latency). As expected, the more popular item listings are able to provide larger anonymity sets without sacrificing review latency. Less popular listings, on the other hand, can be adjusted so that the time to leave a review is not prohibitively long.

Although setting K per item enables more optimal set sizes, it also enables vendors to make insecure choices as they could, for example, create listings where $K = 1$ and completely remove customer anonymity. Under such a design, users would be responsible for only purchasing products offering an anonymity set size they are comfortable with.

C. Review and Item Listing Enumeration

The primary bottlenecks in the enumeration process are the size of the block headers that must be stored, and the number of blocks that need to be downloaded by a resource-constrained user. To evaluate these, and explore the tradeoff between persistent state and block downloads per query, we use, again, the dataset from Soska and Christin [41] to obtain a realistic distribution for item listings and reviews.

As stated in §V-E, a user needs to find the last review for the item to check if there are any reviews missing in the response of the untrusted node. Thus, user is required to download some number of blocks, and the number of blocks that a user downloads depends on the number of times she erroneously believes that the transaction of interest is stored in

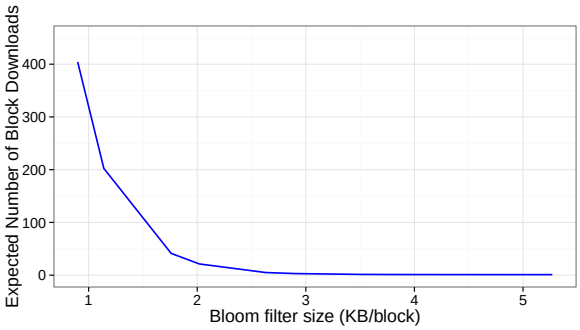


Fig. 4: Expected number of blocks downloaded to enumerate $\mathcal{R}_{(\cdot,i)}$ vs. per-block Bloom filter size.

a particular block due to a false positive in the Bloom filter. The number of these mistakes that she commits depends on (1) the false positive rate of the Bloom filter, which comes at the cost of size and (2) the number of blocks between the transaction she is looking for and the end of $\mathcal{L}$. To test the second point, we inserted 500 random transactions between every transaction in the real dataset to produce a blockchain with 722,242 blocks containing 1,500 transactions each, which is approximately the average number of transactions per block in Bitcoin [2]. We then performed benchmarks by testing only the samples drawn from the real dataset, effectively creating a ledger which is larger than that of Bitcoin at the time of writing and simulating a highly active system where transactions become buried quickly while maintaining the ground truth shape of the distribution.

Figure 4 shows the trade-off between the size of the Bloom filter stored in the block header and the expected number of blocks that the user needs to download when enumerating $\mathcal{R}_{(\cdot,i)}$. When using less than 2KB of storage, users will on average be required to download anywhere from 30 to 100s of blocks per enumeration which may be prohibitively high. At 2.9KB, a user will download less than 3 blocks per enumeration on average, and at 3.7 KB will download less than 1.2 blocks. A user that stores the headers of *all* 722,242 blocks in the system containing, each containing a 2.9 KB Bloom filter, would require about 2 GB of storage. The cost of checking for matches in the filters is quite low, checking the entire blockchain on a single-threaded CPU takes less than 4 seconds.

In practice, a block may contain several different headers comprised of Bloom filters of different sizes so that a user may strike the trade-off that is most optimal for her. A user may also cache the blocks she has downloaded, reducing the number of blocks required in future queries.

VIII. Discussion and Limitations

We next discuss some aspects of Beaver that our design does not fully address, or that present room for improvement.

Anonymity sets and other anonymity metrics. As mentioned previously, we would like the anonymity set to be the full $\mathcal{P}_{(\mathcal{C},i)}$ for each item i. However, if the anonymity sets for the reviews are publicly known and can overlap, it could lead to intersection attacks. For instance, let us assume $\mathbf{r_p}$ has the anonymity set $\mathcal{P}_{(\mathcal{C},i)}$. Now let assume a new payment $\mathbf{p}'$ is added to $\mathcal{P}_{(\mathcal{C},i)}$ to form $\mathcal{P}'_{(\mathcal{C},i)}$, and a review $\mathbf{r_{p'}}$ is added to the ledger with anonymity set of $\mathcal{P}'_{(\mathcal{C},i)}$. In the worst case, $|\mathcal{R}_{(\mathcal{C},i)}| = |\mathcal{P}_{(\mathcal{C},i)}|$ and all of the past transactions were already used for reviews. Trivially, the anonymity set of $\mathbf{r_{p'}}$ is $\{\mathbf{p}'\}$ and the relationship is uniquely identified. We therefore use mutually exclusive anonymity sets to avoid such attacks. In Beaver, creating such sets causes the delay in reviews.

Apart from anonymity sets, there are also other anonymity metrics we could use. Differential privacy [22] in particular has shown a lot of promise in guaranteeing privacy of data in a database while maintaining utility. However, traditional differential privacy usually focuses on protecting the database (in case of a DAM, the ledger) when the users making queries do not have access to the database. This is not the case for Beaver, which makes it hard to apply directly to protect the relationship between payments and reviews. We could potentially use similar ideas to Vuvuzela [44], where noise reviews (i.e., fake reviews) are injected to hide the relationships. Unfortunately, this would significantly increase the size of the ledger and could also lower the utility of the reviews due to the abundance of indistinguishable fake reviews. We believe that anonymity in Beaver, in particular with respect to complicated adversaries that may have partial knowledge of variables is intersting. We would still like to explore stronger forms of anonymity in future work.

Vendor privacy. Though our system offers a high level of privacy and anonymity for buyers, it only offers limited protection for vendors. Beaver provides the ability to hide which items a vendor sells, since it generates a fresh pseudonym for each item; however, the number of transactions is made public via the ledger. This is necessary in Beaver, as customers need explicit information about their anonymity set before leaving a review. One could argue that such transparency and auditability of the vendors may be good for the market as a whole, but this may not be desirable for vendors who want to conceal their transaction volume. We defer this potential improvement to future work.

Unclaimed reviewing fees. When a payment transaction is approved, a value of at least f_{rev} is left in the customer's account to cover the cost of the (future) review. Customers may however elect not to leave reviews for a variety of reasons (lack of interest, loss of cryptographic material...), causing the funds to be effectively orphaned. This can be addressed through a network-wide policy: review fees unused for a long time could be claimed by miners, distributed among the customers in the anonymity set, or locked forever and offset by printing new currency.

Advanced Sybil detection and reputation calculation. Beaver provides users with a large corpus of raw information in the form of the public ledger $\mathcal{L}$. Beaver makes no effort beyond the details of its specification to identify reviews that have been generated as the result of a Sybil attack. Other research [18], [46], [47] uses a variety of heuristics and machine learning techniques to detect Sybil *behavior*. Customers in Beaver can run some of these algorithms themselves over the ledger to make their own decisions. In a similar fashion, customers may in practice run their own algorithms for distilling the richness of review messages and values down to a concise number that can be easilly compared across items.

Customers who deanonymize themselves. In the review

message field, a customer might link her transaction or reveal her true identity, thereby reducing the anonymity set of all other customers who bought the item. Such user errors fall outside the scope of Beaver, but might be addressable through well-designed user interfaces.

Economically irrational adversaries. Beaver aims to protect its users against adversaries who are economically rational. Our notion of credibility, for instance, is derived from the lower bound of the cost to an adversary for generating reviews. This line of defense may fail against economically irrational adversaries, who for instance, may spend $1,000 generating fake reviews to setup a honeypot item listing, or to deanonymize a customer buying a $20 product. Such adversaries are outside the threat model of Beaver. However, large-scale attacks on customer anonymity are harder. Deanonymizing every customer requires that the adversary owns at least $\gamma = K - 1$ transactions for *every* K anonymity set. For unpopular item listings, such popularity surges should be easily detectable on the public ledger. For popular listings, this attack would require a tremendous amount of financial resources, which even economically irrational adversaries may not have.

Collusion attacks. In a collusion attack, a customer or vendor sends a transaction directly to a rogue miner instead of broadcasting it. The miner eventually mines the transaction into the network, and returns some of the awarded fees back to the customer or vendor. Miners may have an incentive to do this for expensive transactions (with a high tax), as they would not compete against other miners; in exchange, the customer or vendor could receive a discount on their transaction. Such an attack undermines review credibility. This attack only works when side-payments are possible: A defense is to simply have the network reject mined transactions of which there exists no prior knowledge. This defense may however be too restrictive unless transactions can be instantaneously (and reliably) broadcast to the entire network. Absent such an ideal broadcast channel, a more permissive defense could be to use a communication protocol, such as Riposte [16], with the ability to reveal messages to the network anonymously.

Advanced searching. The search process we describe in this paper for lightweight user nodes only consists of enumerating all items whose description matches given keywords. To improve usability, one may want more advanced search features, such as date ranges, price ranges, and popularity. Understanding the theoretical limits on the optimal trade-off between enumeration efficiency and the amount of state a user holds is also interesting. We defer the answers to these questions to future work.

Values of fees. The values of the different fees, in particular f_{tax}, plays a crucial role in our system and impact the ability of Beaver to deter Sybil attacks. Unfortunately, optimal fee values are currently unknown, and thus we cannot rule out strange or adversarial behaviors that may rise depending on the values of the fees. However, these values are system-wide (unlike transaction fees in Bitcoin), and we believe that users will not be able to easily take advantage of unoptimal fee values. We leave a more formal, likely game theoretic, analysis to future work.

IX. Related Work

We describe related efforts for establishing reputation in distributed or anonymous settings, and anonymous payments schemes.

Distributed and anonymous review systems. Dingledine et al. [19] explored the idea of using centralized servers to maintain reputation scores for a node's reliability in a decentralized network. Similarly, Gupta et al. [26] used a third party that they called a reputation computation agent to facilitate reputation in a peer-to-peer network. Androulaki et al. [5] proposed using digital cash schemes where peers mint and send reputation coins using a centralized bank. Unlike Beaver, these proposals all require a trusted centralized agent.

Damiani et al. [17] proposed a system where a user queries the reputation of another node in a peer-to-peer network by polling her peers. The EigenTrust [29] algorithm goes a step further and leverages the transitivity of trust, by computing a normalized reputation score based off of a network of trust. While the approach of polling peers in the network does not require a trusted third party, each node's view of the network is biased by its peers, which store reputation information in a public ledger using global consensus.

Bethencourt et al. [7] proposed using a third party, trusted for both privacy and correctness to provide users with monotonic reputation (bad actions cannot be punished). Blömer et al. [8] have suggested using primitives such as zero-knowledge proofs and ring signatures with the assistance of a third party to enable users to leave feedback for each other and query reputation in a privacy preserving way. Both of these approaches require placing trust in a third party, unlike Beaver.

AnonRep [50] uses linkable ring signatures, verifiable shuffles, and homomorphic encryption to build a reputation system where peers can anonymously rate each other, as long as at least one server in a set of powerful servers is honest. Different from Beaver, this system, however, does not protect again Sybil attacks.

Anonymous payments. Early on, Chaum [14] understood that payment schemes would present privacy issues for digital commerce. Practical pseudonymous payment schemes such as Bitcoin [36] have paved the way for adoption of anonymous marketplaces. Since then, several proposals such as Zerocoin [34], Zerocash [6], Mixcoin [12], and Blindcoin [43] have attempted to either fix weaknesses in Bitcoin or replace it altogether with varying security and usability trade-offs.

The anonymity of a customer in a marketplace can only be as strong as the anonymity provided by the underlying payment scheme. While Bitcoin has become the currency of choice in online anonymous marketplaces, several papers (e.g., [33], [40]) have demonstrated that one can cluster users by aggregate behavior, thereby evidencing the weakness of Bitcoin's anonymity guarantees. For this reason, we advocate the use of payment schemes with strong provable security guarantees such as Zerocash [6].

X. Conclusion

We have presented Beaver, a decentralized anonymous marketplace (DAM). Beaver uses anonymous payments, a consensus protocol, zero-knowledge proofs, and linkable ring signatures to instantiate a secure DAM. We showed Beaver preserves the anonymity of customers, and incorporates an incentive structure inspired by existing cryptocurrencies, thereby motivating a healthy network while simultaneously lower-bounding the cost of Sybil attacks.

REFERENCES

[1] Advanced crypto library for the Go language. https://github.com/DeDiS/crypto.

[2] Bitcoin Block Explorer. https://blockchain.info/.

[3] I2P: The internet invisible project. http://www.geti2p.net.

[4] Openbazaar docs, 2016. https://docs.openbazaar.org/.

[5] E. Androulaki and S. Choi. Reputation systems for anonymous networks. In *Proc. PETS'08*, July 2008.

[6] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, and E. Tromer. Zerocash : Decentralized Anonymous Payments from Bitcoin. In *Proc. IEEE S&P*, pages 459–474, May 2014.

[7] J. Bethencourt, E. Shi, and D. Song. Signatures of reputation: Towards trust without identity. In *Proc. Financial Crypto.*, pages 400–407, Jan. 2010.

[8] J. Blömer, J. Juhnke, and C. Kolb. Anonymous and publicly linkable reputation systems. In *Proc. Financial Crypto.*, pages 478–488. Jan. 2015.

[9] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *C. ACM*, 13(7):422–426, 1970.

[10] M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications. In *Proc. ACM STOC*, pages 103–112, May 1988.

[11] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. Kroll, and E. Felten. SoK: Research perspectives and challenges for Bitcoin and cryptocurrencies. In *Proc. IEEE S&P*, pages 104–121, May 2015.

[12] J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. Kroll, and E. Felten. Mixcoin: Anonymity for Bitcoin with accountable mixes. In *Proc. Fin. Crypto.*, pages 486–504. Feb. 2014.

[13] J. Camenisch and M. Stadler. Proof systems for general statements about discrete logarithms. Technical Report 260, ETH Zürich, Mar. 1997.

[14] D. Chaum. Security without identification: Transaction systems to make big brother obsolete. *C. ACM*, 28(70), 1985.

[15] N. Christin. Traveling the Silk Road: A measurement analysis of a large anonymous online marketplace. In *Proc. WWW*, pages 213–224, Rio de Janeiro, Brazil, May 2013.

[16] H. Corrigan-Gibbs, D. Boneh, and D. Mazieres. Riposte: An Anonymous Messaging System Handling Millions of Users. In *Proc. IEEE S&P*, pages 321–338, May 2015.

[17] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Managing and sharing servents' reputations in P2P systems. *IEEE T. Know. Data Engr.*, 15(4):840–854, 2003.

[18] G. Danezis. SybilInfer : Detecting Sybil Nodes using Social Networks. In *Proc. NDSS*, Feb. 2009.

[19] R. Dingledine, N. Mathewson, and P. Syverson. Reputation in P2P anonymity systems. In *Workshop on economics of peer-to-peer systems*, Berkeley, CA, June 2003.

[20] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proc. USENIX Security*, San Diego, CA, Aug. 2004.

[21] J. Douceur. The Sybil attack. In *Proc. Intl. Work. Peer-to-peer Systems*, pages 251–260. Springer, 2002.

[22] C. Dwork. Differential privacy. In *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)*, volume 4052, pages 1–12, Venice, Italy, July 2006. Springer Verlag.

[23] I. Eyal and E. G. Sirer. Majority Is Not Enough: Bitcoin Mining Is Vulnerable. In *Proc. Financial Crypto.*, volume 8437, pages 436–454, 2014.

[24] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Proc. CRYPTO*, pages 186–194, Aug. 1986.

[25] R. Goldberg. Lack of trust in Internet privacy and security may deter economic and other online activities, May 2016. https://www.ntia.doc.gov/blog/2016/lack-trust-internet-privacy-and-security-may-deter-economic-and-other-online-activities.

[26] M. Gupta, P. Judge, and M. Ammar. A reputation system for peer-to-peer networks. In *Proc. ACM NOSSDAV*, pages 144–152, June 2003.

[27] M. Jakobsson and A. Juels. Proofs of work and bread pudding protocols. In *Secure Information Networks*, pages 258–272. Springer, 1999.

[28] Joey2781. Vendor threatened to pay me a "visit". Reddit DarkNetMarkets forum, Sept. 2015. https://www.reddit.com/r/DarkNetMarkets/comments/3ms7pr/vendor_threatened_to_pay_me_a_visit/.

[29] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The Eigentrust algorithm for reputation management in P2P networks. pages 640–651, May 2003.

[30] A. Kwon, D. Lazar, S. Devadas, and B. Ford. Riffle: an efficient communication system with strong anonymity. *Proc. PETS*, 2016(2):1–20, 2016.

[31] J. K. Liu, V. Wei, and D. Wong. Linkable spontaneous anonymous group signature for ad hoc groups. *Information Security and Privacy*, 2108:325–335, 2004.

[32] D. Mayzlin, Y. Dover, and J. Chevalier. Promotional reviews: An empirical investigation of online review manipulation. *American Economic Review*, 104(8):2421–55, August 2014.

[33] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage. A fistful of Bitcoins: characterizing payments among men with no names. In *Proc. ACM/USENIX IMC*, pages 127–140, Barcelona, Spain, Oct. 2013.

[34] I. Miers, C. Garman, M. Green, and A. D. Rubin. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. pages 397–411, May 2013.

[35] A. Molavi Kakhki, C. Kliman-Silver, and A. Mislove. Iolaus: securing online content rating systems. *Proc. WWW*, pages 919–930, May 2013.

[36] S. Nakamoto. Bitcoin: a peer-to-peer electronic cash system, Oct. 2008. Available from http://bitcoin.org/bitcoin.pdf.

[37] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Consulted*, pages 1–9, 2008.

[38] P. Resnick and R. Zeckhauser. Trust among strangers in internet transactions: Empirical analysis of eBay's reputation system. *Adv. Applied Microecon.*, 11:127–157, 2002.

[39] R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In *Proceedings of ASIACRYPT*, pages 552–565, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[40] D. Ron and A. Shamir. Quantitative analysis of the full Bitcoin transaction graph. In *Proc. Financial Crypto.*, pages 6–24, Apr. 2013.

[41] K. Soska and N. Christin. Measuring the longitudinal evolution of the online anonymous marketplace ecosystem. In *Proc. USENIX Security*, pages 33–48, Washington, DC, Aug. 2015.

[42] P. Tsang and V. Wei. Short linkable ring signatures for e-voting, e-cash and attestation. *Information Security Practice and Experience*, pages 48–60, 2005.

[43] L. Valenta and B. Rowan. Blindcoin: Blinded, accountable mixes for bitcoin. In *Proc. Financial Crypto.*, pages 112–126, Jan. 2015.

[44] J. van den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich. Vuvuzela: scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles, SOSP 2015, Monterey, CA, USA, October 4-7, 2015*, pages 137–152, 2015.

[45] L. Von Ahn, M. Blum, N. J. Hopper, and J. Langford. CAPTCHA: Using hard AI problems for security. In *Proc. EUROCRYPT*, pages 294–311, 2003.

[46] G. Wang, S. Barbara, T. Wang, H. Zheng, and B. Y. Zhao. Man vs . Machine : Practical Adversarial Detection of Malicious Crowdsourcing Workers. In *Proc. USENIX Security*, pages 239–254, San Diego, CA, Aug. 2014.

[47] G. Wang, M. Mohanlal, C. Wilson, X. Wang, M. Metzger, H. Zheng, and B. Y. Zhao. Social Turing Tests: Crowdsourcing Sybil Detection. In *Proc. NDSS*, pages 1–14, Feb. 2013.

[48] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Dissent in numbers: Making strong anonymity scale. In *Proc. USENIX OSDI*, pages 179–182, Hollywood, CA, Oct. 2012.

[49] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 2014.

[50] E. Zhai, D. I. Wolinsky, R. Chen, E. Syta, C. Teng, and B. Ford. Anonrep: Towards tracking-resistant anonymous reputation. In *Proc. USENIX NSDI*, pages 583–596, Santa Clara, CA, Mar. 2016.