A Trusted Infrastructure for P2P-Based Marketplaces

Tien Tuan Anh Dinh, Tom Chothia, Mark Ryan
School of Computer Science, University of Birmingham, United Kingdom
{*ttd,tpc,mdr*}*@cs.bham.ac.uk*

Abstract

Peer-to-peer (P2P) based marketplaces have a number of advantages over traditional centralized systems (such as eBay). Peers form a distributed hash table and store sale offers for other peers. A key problem in such a system is ensuring that the peers store and report all sale offers fairly, and do not for instance favor their own offers. We give a solution to this problem based on Trusted Computing, but unlike other approaches we do not measure and restrict all firmware and software running on a peer. Instead, we tie offers to monotonic counters in such a way that any attempt to not report an offer, or report it falsely, will be detected.

1 Introduction

P2P-based marketplaces offer a number of advantages over centralized systems. First, they scales better and has no single point of failure, thanks to the P2P infrastructure. Second, there would be no limitation on the types of items being exchanged in the market. In other words, the P2P infrastructure overcomes censorship problems found in centralized systems. Third, no single point of authority means a monopoly cannot arise.

Our system is based on a Distributed Hash Table (DHT) [12, 14]. In particular, each user (a buyer, a seller or both) runs a node in the overlay. An item being sold in the market is listed at a pre-determined node in the network. The description of the item is used as the key, and the node responsible for that key in the overlay will become the *listing node*. As DHTs support the locating of the listing nodes in a deterministic way they make a better choice for P2P-based marketplaces that unstructured networks. In a traditional marketplace, one item may be sold by many different sellers so it is important for a buyer to find offers from all the sellers (to find the one with lowest price, for example). This feature is not easily implemented in P2P infrastructure with undeterministic (and broadcasting) search.

An essential consequence of using DHTs is that a seller will need to trust the listing node to truthfully store and report his item to any potential buyers. There are economic incentives for the listing node to misbehave in this way. For example, if it gets a percentage of the value of each item sold, it will earn more profit by reporting the items with the highest price. If it is also selling the same item, the rational choice is just to report its own item. Therefore, we believe this is an important problem to be addressed if the P2P-based marketplaces are to be realized.

In our proposed infrastructure, the decision of whether to trust another peer is simple. Two parties engage in a protocol; if it is completed successfully, then one party can trust another. Any misbehavior from one party will result in the protocol being terminated prematurely, and hence will be detected by the other party. In particular, we make use of the *monotonic counters* offered by the Trusted Computing infrastructure (and TPM devices [2]). As these counters can only be read or incremented, we tie sale items to their values so that the peer cannot lie, without being detected.

In related work, Balfe et al. [4] propose a secure identity assignment scheme in which the identity of a peer is bound to the underlying trusted device (TPM). Zhang et al. [13] propose a P2P system that uses Trusted Computing to guarantee that all peers are running the correct software. This requires exact knowledge of all software running on a machine and a trusted, immutable BIOS [2] making it impractical in many cases. It is also vulnerable to alterations made to the computers memory after the software has been checked [5]. Our approach on the other hand only requires the presence of a standard TPM and places no restrictions on the software stack. Independently to us, Levin et al. [8] propose an abstraction of a trusted monotonic counter services that can be used to combat equivocation in distributed systems. The main contributions of this paper are:

1. We present a model of P2P-based marketplace infrastructure and state the problem of trust in this model.
2. We propose an implementation using Trusted Computing, which can be readily implemented using current trusted computing devices like the TPMs.

In the next section, we describe the system model and the problem being tackled. In Section 3, we explain our solution and sketch a proof of correctness. More efficient solutions are proposed in Section 4. Lastly, we discussion future work and open issues.

$\mathcal{P} = \{p_1, p_2, .., p_N\}$	set of peers in the network
$\mathcal{D} = \{d_1, d_2, .., d_M\}$	set of data keywords to be stored
$\mathcal{S} : \mathcal{D} \rightarrow \mathbb{P}(\mathcal{P})$	returns the set of sellers of a given item
$v(p, d) \in \mathbb{R}^+$	price that p ($p \in \mathcal{S}(d)$) set on the item d
$\Delta : \mathcal{D} \rightarrow \mathbb{P}(\mathcal{P})$	returns the destination/replicas set for a given item
$\mathcal{W}^p$	is the set of data items stored at p. In particular: $\mathcal{W}^p = \{(d, s, c) \mid d \in \mathcal{D},\ p \in \Delta(d),\ s \in \mathcal{S}(d),\ c \in \mathbb{R}^+\}$
$\mathcal{W}^p_d$	defined as follows: $\mathcal{W}^p_d = \{(d', s', c') \in \mathcal{W}^p \mid d = d'\}$
$f \in \mathbb{R}^+,\ r \in (0, 1)$	flat-rate and variable payment
$p_i \rightarrow p_j : m$	peer p_i sends the message m to p_j

Table 1. Summary of notations used in the model

2 Model and Problem Description

System Model

The P2P-based marketplace is built on top of a structured overlay [12, 14]. Regarding the overlay, we make three assumptions: (1) the network is static[1] (2) the assignment of peers' identifiers is secure [4] (3) the routing protocol always returns correct identities of the destinations. Table 1 shows notations used in the model. Two main protocols used in the system are:

1. *publish*(d): a seller p ($p \in \mathcal{S}(d)$) executes this by first finding the set of listing nodes $\Delta(d)$ for their item d, relying on the routing protocol of the P2P overlay. p chooses a node in this set $p_d \in \Delta(d)$, and sends the message $p \rightarrow p_d : (d, p, v(d, p))$. The listing node updates its state ($\mathcal{W}^p ::= \mathcal{W}^p \cup \{(d, p, v(d, p))\}$) and p also makes a flat-rate payment of value f to p_d.
2. *retrieve*(d): a buyer p_r ($p_r \notin \mathcal{S}(d)$) executes this protocol by first finding $\Delta(d)$, in same way as *publish*(d). p_r starts a session t with each peer p_d ($p_d \in \Delta(d)$), from which it receives the set of offers ρ_t ($\rho_t \subseteq \{d\} \times \mathcal{S} \times \mathbb{R}$). If $\rho_t \neq \varnothing$, p_r then selects p_s such that $(d, p_s, v) \in \rho_t$ (e.g. by lowest price). The buyer p_r makes the payment of value v to p_s for the item d. If listing node p_d takes a commission at rate r from selling d at then its total profit from the sale will be $r.v + f$.

Problem Description

The problem we are tackling is that listing node p_d would not tell a buyer p_r all the information it stores about d. In other words,

$$\rho_t \neq \mathcal{W}^{p_d}_d$$

There are a number of reasons why this problem arises. First, the variable profit that p_d gets in the transaction t is $r.v$ where $(d, p_s, v) \in \rho_t$. Therefore, if both $e = (d, p_s, v)$ and $e' = (d, p'_s, v')$ are in $\mathcal{W}^{p_d}_d$ and $v > v'$, p_d will be likely to gain more profit by not including e' in ρ_t. Second, if $v(p, d)$ is the same for all $p \in \mathcal{S}(d)$ and $p_d \in \mathcal{S}(d)$, p_d would gain the most profit by sending back $\rho_t = \{(d, p_d, v(p_d, d))\}$. Third, when $p_d \notin \mathcal{S}(d)$, p_d may choose to only include (d, p'_d, v') in ρ_t, where p'_d is colluding with p_d or pays higher fees. In this paper, we propose an implementation of this model that satisfies:

$$\forall d, p, v \bullet\ (d, p, v) \in \rho_t \iff (d, p, v) \in \mathcal{W}^{p_d}_d \qquad (1)$$

This property implies that if *publish*(d) and *retrieve*(d) operations are successful, p_r will get all the information regarding the item d stored at p_d.

[1] We discuss how to relax this in the last section

3 Proposed Solution

Our solution require a number of trusted operations that can be executed at any peer. In particular, we assume an underlying Trusted Computing infrastructure [1], which supports the following features:

- Monotonic counter: only *read* and *increment* operations are permitted. The counter value cannot be reverted.
- Signing key: unique for each peer. Its validity can be checked by any other peers in the system.
- `ReadSign`(n) returns $(t, Sign(read, t, n))$ where t is the latest counter value. The signature $Sign(t, n)$ is generated with the signing key and nonce n.
- `IncSign`(n) returns $(t, Sign(inc, t, n))$. This increments the counter, then signs the latest value using the signing key and nonce n.
- Under the standard Dolve-Yao attacker model, these operations are secure. More specifically, one cannot obtain a valid signature without having executed the `ReadSign` or `IncSign` command.

These features can be realized with the Trusted Platform Module (TPM) [2]. A typical TPM supports at least 4 monotonic counters, accessed via `TPM_ReadCounter` and `TPM_IncrementCounter` commands. The Attestation Identity Key (AIK), generated and protected by the TPM, can be used as the signing key. Signatures are verified using Privacy Certificate Authority (CA) or Direct Anonymous Attestation (DAA) protocol. To implemented `IncSign`(n) the owner: (1) establishes a transport session l with the TPM, using `TPM_EstablishTransport` (2)

wraps `TPM_IncrementCounter` command into *wc* and then executes `TPM_ExecuteTransport`(wc, l) (3) executes `TPM_ReleaseTransportSigned`(l, n). The result of this is a signature on the transport session log (including new counter value). The implementation of `ReadSign` is similar. An AIK is used for signing and n is the non-replaying nonce. These operations are securely executed inside the TPM, hence the signature cannot be faked.

Using `ReadSign` and `IncSign`, we modify *publish* and *retrieve* operations as follows:

- *publish*(d): p first finds $p_d \in \Delta(d)$:
  1. $p \rightarrow p_d : (e = (d, p, v(p, d)))$. p_d executes `IncSign`(SHA1(e)) and updates its state, i.e. $\mathcal{W}^{p_d} = \mathcal{W}^{p_d} \cup \{e\}$.
  2. $p_d \rightarrow p : (t, \sigma_t)$. p verifies that (t, σ_t) is the result of p_d.`IncSign`(SHA1(e)). In the implementation using TPMs, it checks that the σ_t was generated on a log of a transport session, inside which only the `TPM_IncrementCounter` command was executed. p then makes the flat-rate payment of value f to p_d,
- *retrieve*(d): p_r first finds $p_d \in \Delta(d)$:
  1. $p_r \rightarrow p_d : n$. n is a random nonce
  2. $p_d \rightarrow p_r : (t, \sigma_t)$. p_r verifies that (t, σ_t) is the result of p_d.`ReadSign`(n). If successfully, p_r can accept t as the latest counter value of p_d.
  3. $p_d \rightarrow p_r : (w_i, \sigma_i)$ for all $i \in [1, t]$. p_r verifies that σ_i is the correct signature from p_d, i.e. $\sigma_i = Sign(inc, i, \text{SHA1}(w_i))$.
  4. After step 3, p_r has received $U_t = \{w_i \,|\, i \in [1, t]\}$. Then, it can construct ρ_t as below:

$$\rho_t = \{(d', s', v') \in U_t \,|\, d' = d\} \qquad (2)$$

The proof that Eq.1 holds goes as follows. It can be seen that $|W^{p_d}| = |U_t| = t$, as all *publish* operations were completed successful. Furthermore, for all $w_i \in U_t$, it follows that $w_i \in W^{p_d}$. If this is not true, p_d must be able find $w_i' \neq w_i$ such that SHA1(w_i) = SHA1(w_i') (so that the verification of σ_i still succeeds). This contradicts the non-collision property of the SHA1 hash function. Therefore, we have $U_t = W^{p_d}$. Combining with Eq.2 and the definition of $W_d^{p_d}$ in Table 1, we can conclude that Eq.1 holds.

4 More Efficient Solutions

The *retrieve* protocol in Section 3 involves sending (w_i, σ_i) for all $1 \leqslant i \leqslant t$. This will not scale well, as $|\mathcal{W}^{p_d}|$ can be very large. In this section, we describe briefly two implementations which are more efficient. One is probabilistic, i.e. it only guarantees Eq.1 holds with a specified probability π. The other relies on more advanced features of the Trusted Computing infrastructure.

4.1 Probabilistic Solution

In this implementation, p_d builds a *hash tree* on top of $\mathcal{W}^{p_d}$. We use 2-3 Merkle trees [10], in which leaves are ordered and the insertion and update operations only involve nodes in the hash path, i.e. $O(h)$ where h is the height of the (balanced) tree. The hash value of d ($W_d^{p_d} \neq \varnothing$) is stored at a leaf of the tree. The latest root hash is included in σ_t where t is the latest counter value.

To check if the hash tree was constructed correctly at p_d, p_r starts a *challenge - response* protocol. In the literature, such a protocol is used for generating *interactive zero-knowledge* proofs [7]. Due to space constraint, we will not explain this in more details. In *publish*(d), p_r verifies if the new root hash is updated correctly by comparing it with its own calculation on the hash path given by p_d. Because leaves in the tree are ordered, in *retrieve*(d) p_r can efficiently count the number of item d stored in the leaf set, then asks p_d for all the valid tuples of the form (w_i, σ_i) where d is included in w_i. There is a clear trade-off between communication (and computation) overhead and the probability that Eq.1 holds.

4.2 Extra Counters

In the previous implementations, only one monotonic counter is used to *time-stamp* elements in $\mathcal{W}^{p_d}$. Let C be the number of counters available. Having $C > 1$ will effectively reduce the communication and computation overhead to the order of $O(\frac{|\mathcal{W}^{p_d}|}{C})$. If C is large, significant improvement can be achieved.

The current TPM devices support a small number of monotonic counters. It is primarily due to the limited amount of permanent storage and possibly limited computation power. There are a number of approaches that help increase the number of counters. One is to have a small trusted software stack running on top of the TPM (a trusted hypervisor, for example). Another approach is to have hardware extension to the current TPM with USB devices or smart-cards [6]. In our case, we assume a small extension to the current specification of the TPM [11].

More specifically, all counters are organized into a binary Merkle tree [9] with $C = 2^h$ leaves, each of them stores an integer value. The *identifier* of a counter c_{id} is from 0 to $C - 1$ and can be derived from the hash path. The TPM stores value of the root hash r_h in permanent storage, which can only be read by `TPM_ReadHashRoot` and updated by `TPM_IncrementCounterTree`($path$). When the latter is executed, the TPM computes the root hash from the given $path$, which includes a current counter value t_{id}. If the result matches r_h, it updates the leaf with the new value $t_{id} + 1$ as well as other values in $path$. As a consequence, the new root hash is stored at the permanent storage. We will not describe the implementation with TPMs due to the space constraint.

With more counters, peers can now make use of `ReadSign`(c_{id}, n) and `IncSign`(c_{id}, n) operations that read or increment counter c_{id} and produce signatures with a nonce n. They return $(X = (t_{id}, c_{id}), Sign(read, X, n))$ or $(Y = (t_{id}, c_{id}), Sign(inc, X, n))$ where t_{id} is the latest value of counter c_{id}. Assuming C counters at every peer, a data item d will be associated to a counter c_{id}, such that $c_{id} = \text{SHA1}(d)\ module\ C$. c_{id} is included in the messages in step 3 of the *publish*(d) operation as well as in step 2 and 3 of the *retrieve*(d) operation. The peer receiving the message will need to verify, among other things, that c_{id} is the assigned counter for d.

5 Discussion and Future Work

We have presented our early ideas of a trusted infrastructure for P2P market places. We detailed implementations that address the problem of peers not fulfilling requests truthfully. However, there is much room left for future work.

As time passes (t increases) the *retrieve* operation will only send more (irrelevant) data. Even with the improvement proposed in Section 4, because the amount of data grows without bound. We plan to address this issue using *sliding windows*, which can regulate maximum numbers of items stored at any given time.

In this paper, we use the flat-rate payment f as incentive for peers to accept publishing sale items. In practice, it may not be effective enough to stop peers from denying the publish operation and potentially denying a seller from publishing its items. Incentives are also needed to discourage sellers from refusing payment to the publishing peers. In future work, we plan to investigate different incentive schemes (such as utilizing a reputation system) to address this problem.

One of the most important assumptions we made is regarding the underlying Trusted Computing infrastructure. Both the original and the probabilistic solutions could be readily implemented using current TPMs, which come with many machines. One may question whether a large-scale P2P system with all peers having their TPMs switched on is a reasonable assumption. It is partly due to past controversy about the TPM [3] and its being in early stages of development. However, in Section 3, we stressed that our system would not be bound to any specific implementation of the Trusted Computing infrastructure. In particular, any infrastructure supporting the five features listed in section 3 can be used. It could be in the form of smart-cards [8] or online services. If available in large scale, such devices or services could be better choices than TPMs because of their flexibility and wider range of trusted functionalities (for example, support for large number of monotonic counters is a built-in feature).

Our model of the system could be made more realistic in a number of ways. First, we are working on solutions using Trusted Computing to realize the secure routing assumption of structured P2P overlays, under dynamic network conditions. Second, letting p_d remove the item d from $\mathcal{W}^{p_d}$ after the successful transaction between p_r and p_s regarding d would be desirable. The protocols must ensure that p_d could not arbitrarily remove items without being detected. One solution would be to use the monotonic counters to implement a time-out mechanism such that buyers only query items that have not expired.

References

[1] https://www.trustedcomputinggroup.org/home.

[2] https://www.trustedcomputinggroup.org/specs/tpm/.

[3] R. Anderson. Trusted computing FAQ. http://www.cl.cam.ac.uk/ rja14/tcpa-faq.html.

[4] S. Balfe, A. D. Lakhani, and K. G. Paterson. Trusted computing: providing security for peer-to-peer networks. In *Fifth International Conference on Peer-to-Peer Computing*, pages 117–24, 2005.

[5] S. Bratus, N. D'Cunha, E. Sparks, and S. W. Smith. Toctou, traps, and trusted computing. In *Trust '08: Proceedings of the 1st International Conference on Trusted Computing and Trust in Information Technologies*, pages 14–32, 2008.

[6] P. England and T. Tariq. Towards a programmable TPM. In *Trust*, pages 1–13, 2009.

[7] U. Feige, A. Fiat, and A. Shamir. Zero knowledge proofs of identity. In *19th ACM Symposium on Theory of Computing*, pages 210–17, 1987.

[8] D. Levin, J. R. Douceur, J. R. Lorch, and T. Moscibroda. Trinc: Small trusted hardware for large distributed systems. In *USENIX Symposium on Networked Systems Design and Implementation*, 2009.

[9] R. C. Merkle. A certified digital signature. In *Advances in Cryptology*, pages 218–38, 1989.

[10] M. Naor and K. Nissim. Certificate revocation and certificate update. In *7th USENIX Security Symposium*, pages 217–28, 1998.

[11] L. F. Sarmenta, M. van Dijk, C. W. O'Donnell, J. Rhodes, and S. Devadas. Virtual monotonic counters and count-limied objects using a TPM without a trusted OS. In *Workshop on Scalable Trusted Computing*, pages 27–42, 2006.

[12] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup service for internet applications. In *SIGCOM*, pages 149–60, 2001.

[13] X. Zhang, S. Chen, and R. Sandhu. Enhancing data authenticity and integrity in p2p systems. *IEEE Internet Computing*, 9(6):42–49, 2005.

[14] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: an infrastructure for fault-tolerant wide-area location and routing. Technical report, Berkeley, 2001.