# ECPAS: A Blockchain-based E-Commerce Price Auditing System

Toshiki Takakubo*, Ruidong Li*, Haihan Nan†*, Qun Jin‡, Zhou Su§, and Huaming Wu¶.
*Kanazawa University, Kanazawa, Japan
† Xidian University, Xi'an, China
‡Waseda University, Tokorozawa, Japan
§Xi'an Jiaotong University, Xi'an, China
¶Tianjin University, Tianjin, China
Email: *tosiki219@stu.kanazawa-u.ac.jp; liruidong@ieee.org; hhnan@stu.kanazawa-u.ac.jp,
‡jin@waseda.jp, §zhousu@ieee.org, ¶whming@tju.edu.cn

***Abstract*—In recent years, with the widespread of the Internet and further big data, E-Commerce (EC) has emerged as a popular medium for users to engage in online transactions of products and services. Generally, Service Providers (SPs) of EC collect users' personal information and utilize advanced big data technologies to enhance their services. However, the price discrimination problem may also arise based on personalized information, where malicious SPs analyze users' historical orders to provide the same products or services at varying prices depending on their characteristics. In this paper, we propose a price auditing system called E-Commerce Price Auditing System (ECPAS) to resolve this problem. ECPAS consists of four smart contracts: User Registration Contract, Product Registration Contract, Insurance Purchasing Contract, and Price Auditing Contract, which realize EC price auditing and financial compensation for price discrimination based on a private blockchain. Meanwhile, ECPAS utilizes InterPlanetary File System (IPFS) to efficiently store product data. Experimental results demonstrate that ECPAS achieves a higher processing speed of 5 million price auditing per day while maintaining low gas and on-chain storage costs based on the IPFS.**

***Index Terms*—Big Data, Blockchain, Smart contract, IPFS, E-Commerce**

## I. Introduction

With the popularization of E-Commerce (EC) in recent years, billions of users prefer to use EC websites to purchase services and products, which generate a large amount of data about users and corresponding products. Specifically, Service Providers (SPs) can collect users' personal information and order history to provide customized services. By understanding and catering to the preferences of individual users, they enhance the user experience and boost convenience. Unfortunately, the generated big data is difficult to be efficiently analyzed due to the large quantity and different characteristics [1]. Meanwhile, some malicious SPs may unfairly profit by misusing this big data for unethical purposes, such as price discrimination [2].

Price discrimination refers to the sale of charging different prices for the same products or services depending on individual users [3]. For example, a loyal user might be charged a higher price than a new user, or a rich person might be charged more than a poor person. In this paper, we consider such personalized pricing orders as rogue orders. In fact, some price discrimination has been reported. In 2000, Amazon requested a higher price for loyal users based on their cookie record [4]. Besides, there have been reports of price discrimination on some EC websites, varying based on the regions where users live [5]. Therefore, price discrimination indeed exists as a substantial problem.

Price discrimination can lead to various long-term losses for both users and the economy. Generally, many people consider price discrimination as unfair or manipulative [6]. As a result, users shopping on EC websites need to compare prices across different platforms to ensure they are paying reasonable amounts for their purchases. This not only takes time, but also adds unnecessary complexity to what should be a simple shopping experience. If one faces price discrimination, the current solution is to file a lawsuit, which not only is a tedious process but also time-consuming. Furthermore, from a macroeconomic standpoint, it can negatively impact the economy if users lose trust in online sellers due to price discrimination [6].

To address the problem of price discrimination, a price auditing system [7] has been proposed using blockchain and Smart Contract (SC). However, the system proposed in [7] is designed for ride-hailing platforms and would incur substantial storage costs on the blockchain when applied to EC, thus rendering it inefficient. To overcome this challenge, we propose a E-Commerce Price Auditing System (ECPAS). Specifically, ECPAS integrates blockchain, SC, and the InterPlanetary File System (IPFS) to achieve transparent and dependable price auditing without the need for third-party involvement, simultaneously leading to substantial reductions in blockchain storage costs. When a user completes an order on the EC website, ECPAS assesses whether the order is rogue. If it identifies the order as a rogue order, it promptly offers financial compensation to the affected user and imposes a penalty on the SP. Our experimental results demonstrate the superiority and effectiveness of the proposed ECPAS system, which has achieved the first-ever blockchain-based price auditing in the field of EC.

The rest of this paper is organized as follows. Section II

reviews related work about the key components of our system. Section III describes the details of the proposed ECPAS system. Section IV demonstrates the performance evaluation results. Section V concludes this paper.

## II. Related Work

### A. Blockchain

The basic concept of blockchain was initially proposed by S. Nakamoto in 2008 [8]. Blockchain enables peer-to-peer value exchange without a third party. In essence, blockchain serves as a decentralized storage system, which is transparent and tamper-resistant relying on a well-designed data structure and consensus mechanism [9], [10]. The fundamental characteristics of blockchain can be summarized as follows:

1) Decentralization: Blockchain is a decentralized system that runs on a P2P network, which means there is no system downtime and no central authority controls it.
2) Transparency: Every transaction on the blockchain is open for viewing by all participants, thereby creating a system with complete visibility.
3) Security: Blockchain ensures the security of transactions against tampering through the use of intricate cryptography and robust data structures.

The aforementioned characteristics endow blockchain with the ability to enable reliable and transparent price auditing in the proposed ECPAS without a third party.

### B. Smart contract

The term SC was first introduced in the mid-1990s by N. Szabo [11], which refers to a computer program of contracts. Blockchain enables the essence of SC by defining the program. The program defines the execution and response conditions of the contract, allowing for automatic completion without the involvement of a third party. Therefore, SCs can be executed swiftly and autonomously compared to traditional contracts. SC is executable on the SC development platform, Ethereum Virtual Machine [12].

In the context of ECPAS, SCs play a vital role in facilitating price auditing and detecting rogue orders. We can achieve prompt and automated financial compensation by paying financial compensation to affected users and imposing fines to malicious SPs who produce rogue orders.

### C. InterPlanetary File System

IPFS [13] is a distributed file system that runs on a P2P network. IPFS aims to store and distribute data across various network nodes while unifying all computing devices under a single file system. It integrates a distributed hashtable, an incentivized block exchange, and a self-certifying namespace to establish a content-addressed block storage model and hyperlinks, enabling efficient data throughput [13]. IPFS can be applied in various fields, and its utilization can significantly improve system performance [14], [15].

After uploading data to IPFS, a Content IDentifier (CID) is returned. The CID is a unique cryptographic hash string that serves as an exclusive address to reference the data on the network. To access desired data, one needs to obtain the CID and request it from IPFS. Considering that blockchain is not suitable for storing extensive volumes of data, IPFS emerges as an optimal solution by only storing the CID rather than data, which greatly reduce storage space. Since our proposed system need to store a large amount of data, IPFS is utilized to implement price auditing on the blockchain with minimal storage costs.

### D. Price auditing system

Price discrimination is not solely confined to EC but also exists in the realm of ride-hailing services [16]. To solve this problem in ride hailing services, a blockchain and SC based system named Spas has been proposed for price auditing [7]. However, there is a challenge in directly applying Spas for price auditing in EC. Spas is designed for a ride-hailing service context, where the base price registered for price auditing on the blockchain typically includes information such as the price per kilometer. In contrast, EC requires the registration of a vast amount of product data, leading to the issue of excessive storage costs on the blockchain. We solve this problem by using IPFS and thereby realizing efficient price auditing in EC.

## III. ECPAS System Design

In this section, we introduce the framework of ECPAS. First, we provide an overview. Subsequently, we describe the workflow and events of the SC process. Finally, we offer detailed descriptions of key events, including user and product registration, insurance purchase, and price auditing processes.

### A. Overview

ECPAS is a decentralized system that uses blockchain, SC, and IPFS. By using blockchain, which plays a crucial role in decentralized processing, a fair price auditing can be realized without a third party. Meanwhile, the integration of SC enables the automatic detection of rogue orders and facilitates prompt financial compensation. Additionally, ECPAS is able to reduce the storage cost on the blockchain by utilizing IPFS. Fig. 1 shows the overview of ECPAS. First, SPs register their own products on the EC website and store the corresponding product data on IPFS. Then, CID returned from the IPFS is stored on the blockchain. Subsequently, SC is invoked when users purchase products and execute price auditing through the automated program. Finally, the result is stored on the blockchain.

ECPAS is composed of four SCs: User Registration Contract (URC), Product Registration Contract (PRC), Insurance Purchasing Contract (IPC), and Price Auditing Contract (PAC). The URC allows both users and SPs to register in the ECPAS system. The PRC facilitates SPs in registering and updating product information on the blockchain. The IPC enables users to purchase insurance from SPs and terminate it if needed. The PAC executes price auditing each time an order is completed, and achieves financial transactions such as compensation payments when detecting malicious orders.

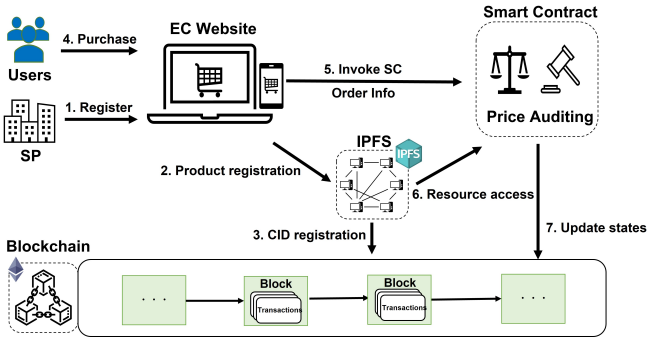



Fig. 1: Overview of ECPAS

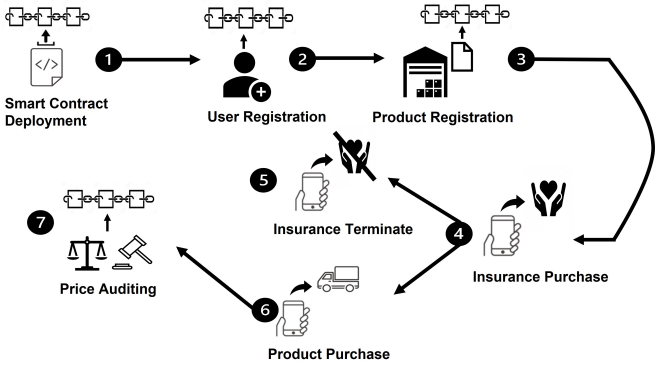



Fig. 2: SC workflow and events

Fig. 2 shows the SC workflow and events. First, the SC is deployed on the blockchain. Then, users and SPs register to participate in ECPAS. Next, SPs register their product data on the blockchain. After that, users purchase insurance from SPs. Only users who have purchased insurance are eligible for price auditing. Finally, when a user completes an order on the EC website, the order is audited. We will introduce details of these events in the sections that follow.

### B. User and SP Registration

Users and SPs wishing to participate in the ECPAS system must register their own information through the URC. The URC includes the userRegister function for user registration and the spRegister function for SP registration.

**userRegister function**

- **Require**:
  - User's account address $\mathrm{Addr_u}$
- **Ensure**: User registration in ECPAS

where $\mathrm{Addr_u}$ is the user's account address for receiving financial compensation and other payments. In addition, $\mathrm{Addr_u}$ is used to uniquely identify the user. The userRegister function employs $\mathrm{Addr_u}$ to verify that the user has not already been registered. Once this is confirmed, the user registration is executed by registering $\mathrm{Addr_u}$ in ECPAS.

**spRegister function**

- **Require**:
  - SP's account address $\mathrm{Addr_{sp}}$
  - Deposit sent to contract address $Deposit$
- **Ensure**: SP registration in ECPAS

where $\mathrm{Addr_{sp}}$ is the SP's account address. It is used for identification and financial transactions, similar to user addresses. The spRegister function employs $\mathrm{Addr_{sp}}$ to verify that the SP has not already been registered. Once confirmed, the SP registration is executed by registering $\mathrm{Addr_{sp}}$ in ECPAS. In addition, the SP needs to pay $Deposit$ to the contract address. The $Deposit$ is treated as a reserve against fines incurred when an SP engages in price discrimination. This ensures that the contract always has sufficient funds, thereby eliminating the risk of unpaid fines.

### C. Product Registration and Update

In order to reference a base price of products for price auditing, SP needs to register the latest product data through PRC. SP needs to store the product data on IPFS. By leveraging the IPFS, the CID is received and stored on the blockchain. Essentially, the product data is stored on the IPFS, and only the CID is stored on the blockchain, reducing the storage cost on the blockchain. The price stored on the IPFS is defined as the fair selling price, and any order exceeding that price is considered a rogue order. The PRC has the productRegister function to register products and the productUpdate function to update product data.

**productRegister function**

- **Require**:
  - SP's account address $\mathrm{Addr_{sp}}$
  - Product reference address on IPFS $cid$
- **Ensure**: Register $cid$ on blockchain

where $cid$ is a product reference address returned after registering the product data on IPFS. The productRegister function checks if SP is already registered and if the product is not already registered, using $\mathrm{Addr_{sp}}$ and $cid$, respectively. If the conditions are met, the function registers the product's reference address, $cid$, on blockchain.

**productUpdate function**

- **Require**:
  - SP's account address $\mathrm{Addr_{sp}}$
  - Original product reference address $cid$
  - New product reference address $newCid$
- **Ensure**: Update $cid$ to $newCid$ on blockchain

The productUpdate function employs $\mathrm{Addr_{sp}}$ and $cid$ to verify whether the SP and the product are already registered. If these conditions are met, $cid$ is updated to $newCid$.

### D. Insurance Purchase and Termination

ECPAS introduces an insurance mechanism to limit the use of the blockchain network's computing resources and provide financial incentives to SPs that do not engage in price discrimination. Given the limited computing resources of the blockchain and the large number of EC users, performing a price auditing for all users would significantly increase

the computational overhead and potentially render the system inefficient. Therefore, only users who purchase insurance from an SP through IPC are eligible for price auditing. To receive a price auditing, users must incur a certain cost, which can limit the number of price auditing. The IPC includes an insurancePurchase function for the user to purchase insurance and an insuranceTerminate function for the user to terminate the insurance.

**insurancePurchase function**

- **Require**:
  - User's account address $\mathrm{Addr_u}$
  - SP's account address $\mathrm{Addr_{sp}}$
  - Insurance expiration date $period$
  - Insurance payments to contract address $f_{ins}$
- **Ensure**: User purchases insurance from SP

where $period$ is the expiration date of the insurance and is defined in units of days. Let $insPerDay$ is the daily insurance payout, then $f_{ins} = insPerDay \cdot period$ is the insurance payout paid by the user. Insurance payments are executed by transferring money to the contract address when the user executes the insurancePurchase function.

**InsuranceTerminate function**

- **Require**:
  - User's account address $\mathrm{Addr_u}$
  - SP's account address $\mathrm{Addr_{sp}}$
- **Ensure**: user terminates insurance from SP

When the insuranceTerminate function is executed, the termination fee $f_{ter}$ is transferred from the contract to the user and SP. Termination fee for the user is defined as $f^u_{ter} = \alpha \cdot f_{ins}$. On the other hand, the termination fee for SP is defined as $f^{sp}_{ter} = f_{ins} - f^u_{ter}$. Let $\alpha$ $(0 \leq \alpha \leq 1)$ is the percentage of the remaining insurance expiration time.

### *E. Price Auditing*

This section explains the details and specific processes of price auditing. We will discuss price auditing algorithm, cash flow and payout settings.

*1) Price auditing algorithm:* When an insured user completes an order on the EC website, the price auditing process begins. First, order information is obtained for both the user and the SP. This information includes the user's and the SP's addresses, the charged price, and the CID. Then, the CID is used to obtain the product's price stored on the IPFS. It is considered the correct price on which to base the price auditing. The order information and price stored on IPFS are used as parameters and passed to the PAC. Then, the SC program is executed. The algorithm for price auditing is shown in Algorithm 1.

First, the necessary information for price auditing is obtained from the order information. The validity of the user's insurance and the registration of the $cid$ are checked. Then, by comparing the hashes of the order information of the user and the SP, it is verified that the order data has not been tampered with (lines 1-6). Once confirmed, the user charged price is derived and judge whether it is a rogue order (line 7-8). If the order is determined to be rogue, the user will be compensated and the SP will be charged a fine. The compensation $f_{Com}$ paid to the user is calculated at three times the user's charged price or 0.05 Ether, whichever is greater. The $f_{Com}$ is paid to the user from the contract, and the SP pays the fine by subtracting $f_{Com}$ from the deposited value. In addition, the user's insurance is terminated at the same time (lines 9-14).

**Algorithm 1** Price Auditing

**Require:**
1) $Order_u$: order's information from user
2) $Order_{sp}$: order's information from SP
3) $price$: price stored on IPFS

**Ensure:** Determine if a price is fair

**Procedure**: Price Auditing Process

1: $\mathrm{Addr_u} \leftarrow$ get user's account address from $Order_u$
2: $\mathrm{Addr_{sp}} \leftarrow$ get SP's account address from $Order_u$
3: $cid \leftarrow$ get product reference address from $Order_u$
4: Check the validity of the user's insurance through $\mathrm{Addr_u}$ and $\mathrm{Addr_{sp}}$
5: Check the validity of the product registration through $cid$;
6: **if** $Hash(Order_u) == Hash(Order_{sp})$ **then**
7: $price' \leftarrow$ get charged price from $Order_u$
8: **if** $price < price'$ **then**
9: $\mathrm{Addr_c} \leftarrow$ get contract address
10: $f_{Com} \leftarrow$ Calculate compensation payout
11: Send $f_{Com} + f^u_{ter}$ from $\mathrm{Addr_c}$ to $\mathrm{Addr_u}$
12: Send $f_{ins} - f^u_{ter}$ from $\mathrm{Addr_c}$ to $\mathrm{Addr_{sp}}$
13: Subtract $f_{Com}$ from the value of the SP deposit
14: Validity of the user's insurance $\leftarrow$ false
15: **else**
16: end process.
17: **end if**
18: **else**
19: end process.
20: **end if**

*2) Cash Flow and Payout Settings:* ECPAS events also produce a series of cash flow, which is summarized in the Table I. Moreover, Table II shows the reward in the price auditing. where U: User, S: SP, and C: Contract. Rewards for price auditing vary considerably depending on whether rogue orders exist or not. On the one hand, when rogue orders do not exist, all user insurance payouts are paid to SPs that do not engage in price discrimination. On the other hand, when rogue orders do exist, the user's reward must be positive and the SP's reward must be negative. In other words, $f_{ins} < f_{Com} + f^u_{ter}$ must be satisfied. If we set $f_{ins} < f_{Com}$, then $f_{ins} < f_{Com} + f^u_{ter}$ always holds because $f^u_{ter}$ is always positive. Furthermore, $Deposit$ must be sufficiently larger than $f_{Com}$ so that the following relationship holds.

$$f_{ins} < f_{Com} < Deposit \tag{1}$$

When implementing ECPAS, it is necessary to set parameters that satisfy this relationship.

TABLE I: Event Cash Flow

| Events | From | To | Amount |
|---|---|---|---|
| spRegister | S | C | $Deposit$ |
| insurancePurchase | U | C | $f_{ins}$ |
| insuranceTerminate | C | U | $f^u_{ter}$ |
| | C | S | $f_{ins} - f^u_{ter}$ |
| report Rogue Order | C | U | $f_{Com} + f^u_{ter}$ |
| | C | S | $f_{ins} - f^u_{ter}$ |
| | S | C | $f_{Com}$(Subtract from deposit) |

TABLE II: Reward for Price Auditing

| Entity | No rogue order existed | Rogue order existed |
|---|---|---|
| U | $-f_{ins}$ | $-f_{ins} + f_{Com} + f^u_{ter}$ |
| S | $f_{ins}$ | $f_{ins} - f_{Com} - f^u_{ter}$ |
| C | 0 | 0 |

## IV. Performance evaluation

In this section, we first introduce our experimental environment. Next, we evaluate the performance of the proposed ECPAS in terms of three indices: the gas cost, the processing speed, and the On-chain Storage Cost (OSC). Finally, we discuss the evaluation results.

### A. Experimental environment

In this subsection, we introduce the experimental environment used for testing the proposed ECPAS. We employ Truffle as the development tool, which is based on the Ethereum solidity language and offers a testing framework and pipeline for the Ethereum blockchain. Ethereum and IPFS provide JavaScript API libraries, named web3.js and ipfs.js, respectively. These libraries furnish a comprehensive set of JavaScript objects and functions that enable interaction with blockchain, SC, and IPFS. We use the Ethereum private blockchain created with a client application called geth. ECPAS is deployed on this Ethereum private blockchain and tested on the WSL2 Ubuntu 20.04.5 operating system using Solidity v0.8.7 language. For the experiment, we create 20000 accounts on the blockchain, each with a balance of 100 Ether. We have prepared a data set of products, each of which has attributes such as product id, product name, and price. This product data is then uploaded to IPFS, generating 10000 CIDs. Considering equation 1, we set $insPerDay$ to 0.00005 Ether, $f_{Com}$ to 0.05 Ether and $Deposit$ to 10 Ether.

### B. Gas cost result and analysis

SC deployment and invocation on the Ethereum blockchain requires some fee cost, called gas, which is introduced to pay compensation to miners and to avoid unnecessary consumption of computing resources and network abuse. For example, the addition of two variables requires 3 gas, while multiplication requires 5 gas [12]. Gas can be paid using the native Ethereum currency named Ether. To calculate the actual fee cost from gas, multiply gas by the gas price. In this paper, the gas price is calculated as 12.5 Gwei, where 1 Ether equals $10^9$ Gwei. In order to evaluate the usability performance of the proposed ECPAS in terms of gas cost, we measure and evaluate the gas required to deploy and call functions.

TABLE III: Gas cost for each function and deployment

| Functions | Gas | Ether |
|---|---|---|
| userRegister | 45623 | 0.000570 |
| spRegister | 90353 | 0.00113 |
| productRegister | 51988 | 0.000650 |
| productUpdate | 56372 | 0.000705 |
| insurancePurchase | 120241 | 0.00150 |
| insuranceTerminate | 53181 | 0.000665 |
| Audit(correct Order) | 39397 | 0.000492 |
| Audit(rogue Order) | 72228 | 0.000903 |
| Contract deployment | 2929051 | 0.0366 |

Table III displays the results of the gas cost. As the audit function is the most frequently executed function in the system, we need to discuss it more. Due to additional processing requirements for rogue orders, we have categorized the results into correct orders and rogue orders. When auditing correct orders, the gas is the smallest compared to the other functions. On the other hand, when auditing rogue orders, the user compensation significantly exceeds the gas cost. Therefore, the gas cost is not considered to be a burden on users in price auditing. From the perspective of the SPs, the insurance mechanism allows them to generate much higher profits than the gas costs. In summary, gas costs are not a burden for either users or SPs.

### C. Processing speed result and analysis

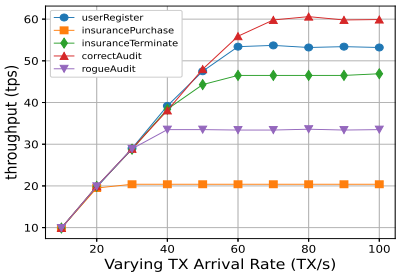
The processing speed of ECPAS is evaluated using SC's transactions per second (tps). Tps is defined by the following equation [17].

$$tps = \frac{Number\ of\ transactions\ per\ block}{Block\ Period} \tag{2}$$

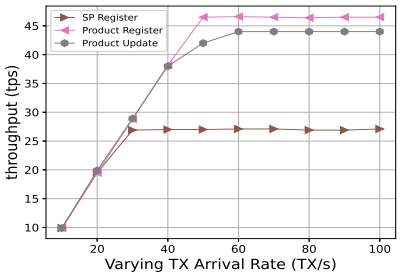
In the Ethereum blockchain, the number of transactions per block depends on the block gaslimit. The block gaslimit is set to 30000000 and the block period is set to 12 seconds, which refers to the value of Ethereum main network [18]. Tps measurements are made using the Caliper benchmarking tool provided by Hyperledger. The measurement is made by executing the each SC function and sending transactions to the blockchain at a fixed rate. The sending rate was varied from 10 to 100. Fig. 3 shows the results of tps measurements. Focusing on the function that audits the correct price, which is the most performed in this system, the maximum tps is about 60. This value means that approximately 5 million price auditing can be executed per day. Taking into account that the number of price auditing is limited by the insurance mechanism, we believe this processing speed is a practical enough number in the real world.

### D. On-chain storage cost result and analysis

OSC is the storage cost of storing data on the blockchain. According to Ethereum's yellow paper [12], the gas required to store a single 256-bit word is 20000. Also, each transaction requires a minimum base cost of 21000 gas. Thus, it is possible to estimate the OSC of each transaction by using the consumed gas [19]. To evaluate the extent to which OSC can




(a) user side tps




(b) sp side tps

Fig. 3: Tps varies by transaction arrival rate

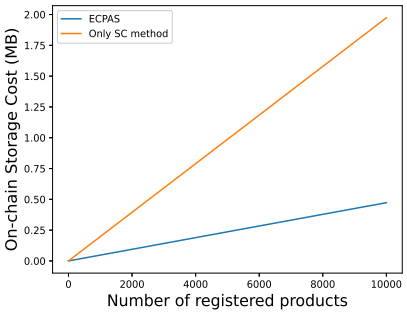



Fig. 4: OSC: ECPAS vs only SC method

be minimized when registering products by utilizing IPFS, we compare the OSC in scenarios involving ECPAS with those involving only SC method [7]. Specifically, with ECPAS, we estimated OSC by utilizing the gas in the productRegister function, as detailed in Table III. In the case of the only SC method, we created an SC for direct storage of product data and estimated the storage cost. The product data includes the following attributes: product ID, product name, price, supplier name, and update date.

The OSC results, depicted in Fig. 4, demonstrate that ECPAS is more efficient and maintains a lower OSC compared to the only SC method. In this experiment, for the sake of simplicity, product data includes only five attributes. However, in a real-world scenario, data would likely comprise a greater number of attributes, and the only SC method could lead to an increase in OSC. Conversely, with ECPAS, the OSC remains unaffected by the number of attributes. Therefore, in a real-world context, we can conclude that ECPAS achieves a lower OSC compared to the existing only SC method.

## V. Conclusion

In this paper, we propose a price auditing system, ECPAS, to solve the problem of price discrimination in EC. ECPAS is a decentralized system utilizing blockchain and SC, which can realize transparent and reliable price auditing without a third party while achieving automatic price auditing and prompt financial compensation. We describe the entire process of the system, including the price auditing algorithm and the insurance mechanism. We implement ECPAS on the Ethereum private blockchain and evaluate its performance. Experimental results demonstrate that ECPAS has a processing speed of 5 million price auditing per day while keeping gas and on-chain storage costs low by using IPFS. We believe these results make ECPAS a feasible solution in the EC. In our forward-looking vision for future work, we believe that with the privacy and security enhancements provided by permissioned blockchain and the scalability improvements provided by sharding, our proposed framework can be extended to other domains such as general online services in the future.

## References

[1] C. Zhang, P. Patras, and H. Haddadi, "Deep Learning in Mobile and Wireless Networking: A Survey," in *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2224-2287, 2019.

[2] A. Hannak, G. Soeller, D. Lazer, A. Mislove, and C. Wilson, "Measuring Price Discrimination and Steering on E-commerce Web Site," *Internet Measurement Conference*, pp. 305–318, 2014.

[3] A. Acquisti, "Price Discrimination, Privacy Technologies, and User Acceptance," In *Proc. CHI Workshop Personalization Privacy*, pp. 1–3, 2006.

[4] "Amazon's old customers 'pay more'," *British Broadcasting Corporation News*, 2000. Accessed: May 15, 2023. [Online]. Available: http://news.bbc.co.uk/2/hi/business/914691.stm

[5] J. Mikians, L. Gyarmati, V. Erramilli, and N. Laoutaris, "Crowd-assisted search for price discrimination in e-commerce: first results," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pp. 1-6, 2013.

[6] F. Z. Borgesius and J. Poort, "Online Price Discrimination and EU Data Privacy Law," *J. Consum. Policy*, vol. 40, no. 3, pp. 347–366, 2017.

[7] Y. Lu, Y. Qi, S. Qi, Y. Li, H. Song, and Y. Liu, "Say No to Price Discrimination: Decentralized and Automated Incentives for Price Auditing in Ride-Hailing Services," in *IEEE Transactions on Mobile Computing*, vol. 21, no. 2, pp. 663-680, 2022.

[8] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Whitepaper*, 2008.

[9] S. Ruoti, B. Kaiser, A. Yerukhimovich, J. Clark, and R. Cunningham, "Blockchain technology: what is it good for?," *Communications of the ACM*, vol. 63, no. 1, pp. 46–53, 2019.

[10] C. Catalini and J. S. Gans, "Some simple economics of the blockchain," *Communications of the ACM*, vol. 63, no. 7, pp. 80–90, 2020.

[11] N. Szabo, "Smart Contracts: Building Blocks for Digital Markets," 1996. Accessed: May 19, 2023. [Online]. Available: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html

[12] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.

[13] J. Benet, "IPFS - Content Addressed, Versioned, P2P File System," *arXiv*, 2014.

[14] S. Zimmermann, J. Rischke, J. A. Cabrera, and F. H. P. Fitzek, "Journey to MARS: Interplanetary Coding for relieving CDNs," *IEEE Global Communications Conference (GLOBECOM)*, pp. 1-6, 2020.

[15] P. Kumar, R. Kumar, S. Garg, K. Kaur, Y. Zhang, and M. Guizani, "A Secure Data Dissemination Scheme for IoT-Based e-Health Systems using AI and Blockchain," *IEEE Global Communications Conference (GLOBECOM)*, pp. 1397-1403, 2022.

[16] A. Mahdawi, "Is your friend getting a cheaper Uber fare than you are?," *The Guardian*, 2018. Accessed: May 15, 2023. [Online]. Available: https://www.theguardian.com/commentisfree/2018/apr/13/uber-lyft-prices-personalized-data

[17] F. Leal, A.E. Chis, and H. González–Vélez, "Performance Evaluation of Private Ethereum Networks," in *SN Computer Science*, vol. 1, no. 285, pp. 1-17, 2020.

[18] "Etherscan - Ethereum (ETH) Blockchain Explorer," Accessed: May 15, 2023. [Online]. Available: https://etherscan.io/

[19] F. Chen, J. Wang, C. Jiang, T. Xiang, and Y. Yang, "Blockchain Based Non-repudiable IoT Data Trading: Simpler, Faster, and Cheaper," *IEEE Conference on Computer Communications (INFOCOM)*, pp. 1958-1967, 2022.