AGChain: A Blockchain-based Gateway for Trustworthy App Delegation from Mobile App Markets

MENGJIE CHEN[*], Mask Network, China
XIAO YI[†], The Chinese University of Hong Kong, China
DAOYUAN WU[‡], The Hong Kong University of Science and Technology, China
JIANLIANG XU, Hong Kong Baptist University, China
YINGJIU LI, University of Oregon, USA
DEBIN GAO, Singapore Management University, Singapore

The popularity of smartphones has led to the growth of mobile app markets, creating a need for enhanced transparency, global access, and secure downloading. This paper introduces AGChain, a blockchain-based gateway that enables trustworthy app delegation within existing markets. AGChain ensures that markets can continue providing services while users benefit from permanent, distributed, and secure app delegation. During its development, we address two key challenges: significantly reducing smart contract gas costs and enabling fully distributed IPFS-based file storage. Additionally, we tackle three system issues related to security and sustainability. We have implemented a prototype of AGChain on Ethereum and Polygon blockchains, achieving effective security and decentralization with a minimal gas cost of around 0.0028 USD per app upload (no cost for app download). AGChain also exhibits reasonable performance with an average overhead of 12%.

CCS Concepts: • **Security and privacy** → **Domain-specific security and privacy architectures**.

Additional Key Words and Phrases: Blockchain, Smart contract, Ethereum, IPFS, App security

ACM Reference Format:
Mengjie Chen, Xiao Yi, Daoyuan Wu, Jianliang Xu, Yingjiu Li, and Debin Gao. 2024. AGChain: A Blockchain-based Gateway for Trustworthy App Delegation from Mobile App Markets. *Distrib. Ledger Technol.* 1, 1 (December 2024), 23 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The increasing popularity of smartphones worldwide is driven by the wide range of feature-rich mobile apps available in various app markets. In addition to official marketplaces like Google Play and the App Store, third-party app markets such as Amazon AppStore and Baidu Market have emerged as significant supplements to official app markets. These third-party markets offer a greater variety of app options for Android users and have gained popularity, particularly in China due to restrictions imposed by the Great Firewall. However, users' growing concerns regarding

[*]The work by Mengjie Chen and Daoyuan Wu were done while at The Chinese University of Hong Kong.
[†]Mengjie Chen and Xiao Yi are the co-first authors.
[‡]Daoyuan Wu is the corresponding author. Contact email: dywu@ie.cuhk.edu.hk.

Authors' addresses: Mengjie Chen, Mask Network, Shanghai, China; Xiao Yi, The Chinese University of Hong Kong, Hong Kong SAR, China; Daoyuan Wu, The Hong Kong University of Science and Technology, Hong Kong SAR, China; Jianliang Xu, Hong Kong Baptist University, Hong Kong SAR, China; Yingjiu Li, University of Oregon, Oregon, USA; Debin Gao, Singapore Management University, Singapore, Singapore.

ACM 2769-6472/2024/12-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

security and privacy have created a demand for permanent, distributed, and secure app access in both official and third-party app markets (more details are available in §2):

D1: *Permanent or more transparent app access.* Firstly, there is a need for enhanced transparency and permanence in app access. Currently, app markets often lack access to older versions of apps, and developers have the freedom to delist apps at their discretion. For example, our analysis of 8,359 popular apps downloaded in November 2018 and November 2019 [51] demonstrates that 13.7% (1,146) of these apps were no longer available after just one year. This emphasizes the demand for permanent or more transparent app access. Presently, users must resort to ad-hoc backup solutions [19, 20] to address this issue.

D2: *Global and distributed access.* Secondly, there is a growing demand for global and distributed access to mobile apps. Currently, many apps on Google Play and the App Store are limited to certain countries, hindering their availability worldwide. Moreover, Google Play itself is subject to censorship in certain regions, resulting in restricted app access for users in those affected areas.

D3: *Secure access.* Thirdly, our measurement shows that a considerable number of third-party Android app markets lack secure app downloading mechanisms. For instance, half of the top 14 Chinese app markets, including popular platforms like Baidu and 360, download apps through insecure HTTP connections. Furthermore, even in markets where secure downloading is available, they generally lack app repackage checking [34, 36, 59]. This poses a significant risk to users' apps, as their integrity and security could be compromised.

While individual measures such as VPN usage (for D2) and HTTPS implementation (for D3) may[1] address some of the demands, they do not adequately resolve the fundamental limitation in terms of scientific research: *the lack of trustworthy app access.* In this paper, we present a novel architecture that integrates the strengths of traditional IT infrastructure with decentralized blockchain technology. Our proposed solution is AGChain (App Gateway Chain), a blockchain-based gateway that connects end users with existing app markets. AGChain provides users with a secure, permanent, and distributed method of app delegation, ensuring trustworthy app downloads. With AGChain, users can opt for indirect app downloads, mitigating concerns related to direct downloads from existing markets. While existing markets continue to provide services, with the exception of delisted apps[2], users can delegate app downloads by inputting the market or custom app URL into AGChain. AGChain then securely retrieves the app from its original market, uploads the raw app file to decentralized storage, and stores the app file index and metadata on the blockchain for future direct downloads through AGChain. Once delegated, the app can be permanently and distributedly downloaded from AGChain by the user and other users.

We utilize smart contracts [26] for implementing our logic on the blockchain and leverage IPFS (Interplanetary File System) [25] for decentralized storage. However, rather than employing them in a conventional manner like other blockchain- and IPFS-based systems [38, 41, 42, 47], we address and overcome two previously neglected challenges:

- Firstly, we significantly reduce gas costs by introducing a series of design-level mechanisms, in contrast to code-level gas optimizations [22, 28, 30, 48]. These mechanisms ensure efficient app storing and retrieval on the blockchain while minimizing gas consumption. Notably, we achieve this by transitioning from conventional in-contract data structures to transaction

[1]Considering that the majority of general users do not have access to paid VPN services and the security status of Chinese app markets is unlikely to change quickly, relying on these technical means may not be as effective as initially assumed.
[2]In addition to the apps available on existing markets, AGChain allows users to upload custom apps through a GitHub URL, such as https://github.com/agchain/agchain/blob/main/Test/A.apk. This feature is specifically designed to accommodate apps that may require payment or could potentially be delisted in the future.

log-based contract storage, resulting in a gas reduction by a factor of 53 per operation (20,000 vs. 375 Gas).

- Secondly, we surprisingly found that IPFS is not inherently distributed. By default, files are only stored in the original IPFS node and cached at IPFS gateways when there are requests. To achieve true distribution in IPFS, we establish an IPFS consortium network that periodically caches app files at IPFS gateways and performs timely backups of apps at crowdsourced server nodes. Additionally, we propose a mechanism to identify fast and uncensored IPFS gateways for the distributed downloading of apps.

To make AGChain secure and sustainable, we further address three AGChain-specific system issues. Firstly, to securely retrieve apps from existing app markets without a network security guarantee, we extract and validate checksums that may be embedded in apps' market pages. In cases where checksums are unavailable, we implement alternative security measures. Secondly, to prevent repackaged apps from polluting our market, we propose a lightweight yet effective app certificate field mechanism to detect such apps. We validate this mechanism experimentally using 15,297 pairs of repackaged apps. Lastly, to incentivize crowdsourced server nodes, we design a mechanism that charges upload fees, ensuring the self-sustainability of the platform.

We have implemented a prototype of AGChain using the widely-used Ethereum blockchain [26] and its layer-2 network, Polygon [16]. The implementation comprises 2,485 lines of code written in Solidity, Python, Java, and JavaScript. In our evaluation, we empirically demonstrate the security effectiveness of AGChain, successfully preventing man-in-the-middle and repackaging attacks on our app delegation. We also assess its decentralization by discovering IPFS gateways in over 21 different locations worldwide. Additionally, we conduct experimental measurements of performance and gas costs. On average, AGChain introduces a 12% performance overhead in tests involving 200 apps from seven app markets. The cost of app upload is approximately 0.002821 Matic (equivalent to 0.0028 USD when one Matic is valued at around 1 USD). We also introduce a batch upload mechanism that reduces gas costs by a factor of 2.02 (for a batch of 10 uploads) to 2.65 (for a batch of 100 uploads). Notably, AGChain does *not* require any gas for app downloads as they do not alter the contract state.

To summarize, this paper presents the following contributions:

- A novel blockchain-based gateway design (§3): We introduce a gateway that facilitates trustworthy app delegation, providing permanent, distributed, and secure access to the apps stored in existing markets (and custom apps upload via GitHub URLs). Our idea of combining traditional IT infrastructure with decentralized blockchain technology opens a new door for future advancements in blockchain systems.
- Addressing previously neglected challenges (§3 and §4): We propose mechanisms to achieve gas-efficient smart contracts and a distributed IPFS design. Furthermore, we overcome three specific system issues unique to AGChain.
- Implementation and extensive evaluation (§5 and §6): We implement a prototype of AGChain on the Ethereum blockchain and its layer-2 network, Polygon. Through extensive evaluation, we assess the performance, gas costs, security, and decentralization of AGChain, validating its effectiveness in these areas.

Availability. The source code of AGChain has been released on https://github.com/VPRLab/AGChain, facilitating its reuse by future blockchain systems.

2 MOTIVATION AND BACKGROUND

In this section, we motivate the need of permanent, distributed, and secure app access by measuring the status quo of existing app markets from §2.1 to §2.3. We also provide the necessary background information in §2.4 to comprehend our blockchain-based design.

2.1 On the Need of Permanent Access by Measuring Delisted Apps on Google Play

It is unclear how many uploaded apps were once delisted. To estimate this percentage, we conducted a specific measurement on a set of 8,359 popular apps collected from Google Play in November 2018 [51]. These apps had one million installs each. After one year, in November 2019, we re-crawled the apps in the same country and discovered that a significant portion, as high as 13.7% (1,146 apps), had been delisted during that time period. This highlights the seriousness of delisted apps in existing app markets and emphasizes the need for a permanent app access mechanism to cater to users who require access to those delisted apps. AGChain provides users with the ability to leverage its delegation to permanently store an app on the blockchain before it is delisted.

2.2 On the Need of Distributed Access by Analyzing Apps with Limited Global Access

An easy observation is that many apps on Google Play and the App Store are limited to specific countries. For example, the TVB media app [7] is restricted to Hong Kong, while the popular Hulu app [10] is available only in the US and Japan. Additionally, there are limitations in accessing English-based apps in Chinese app markets, and vice versa. Although these restrictions may be reasonable from the developers' perspective, many users, particularly those traveling or seeking foreign apps, actively search for methods to bypass these limitations. They resort to online tutorials that involve switching their iTunes accounts to different countries [2] or using VPNs to circumvent Google Play's restrictions [4]. It is important to note that the problem of limited global app access is further compounded by network-based censorship, where certain regions impose restrictions on Google Play [9], denying access to affected users. While we acknowledge the legitimacy of these various controls, it is essential to develop a mechanism for distributed app access as an alternative solution for users to choose.

2.3 On the Need of Secure Access by Measuring Insecure App Downloads in Third-Party App Markets

An unexpected observation is that not all app markets provide secure app downloads as Google Play and the App Store. Surprisingly, we discovered that half of the top 14 Chinese Android app markets still utilize insecure app downloading via HTTP. This opens the door for potential injection of repackaged apps [59], particularly when users connect to public Wi-Fi networks [6] or fall victim to network hijacking [32]. This poses a severe security risk, especially considering that around one billion Internet users in China rely on third-party app markets due to the ban on Google Play.

Table 1 presents our measurement results for the top 16 Chinese Android app markets, as identified by a recent study on app markets [45]. Specifically, we examined whether these markets employ HTTPS as the medium for hosting app downloads. It is important to note that while some market websites may use HTTPS for hosting web content, they may still provide app downloads exclusively through insecure HTTP. To distinguish this situation, we focused solely on app downloading traffic. As indicated in Table 1, half of the 14 Chinese app markets (excluding two markets that do not offer web-based app downloading) utilize insecure HTTP downloading. These markets include prominent Internet giants (e.g., Baidu) and smartphone hardware vendors (e.g., Meizu and Lenovo), as well as specialized app markets (e.g., Anzhi and App China).

Table 1. The measurement result of app downloading security in the top 16 Chinese Android app markets [45].

Market Name	Company Type	Secure App Downloading?
Tencent Myapp	Web Co.	HTTPS ✔
Baidu Market	Web Co.	HTTP ✗
360 Market	Web Co.	HTTP ✗
OPPO Market	HW Vendor	No Website Download
Huawei Market	HW Vendor	HTTPS ✔
Xiaomi Market	HW Vendor	HTTPS ✔
Meizu Market	HW Vendor	HTTP ✗
Lenovo MM	HW Vendor	HTTP ✗
HiApk	Specialized	Cannot Find the Website
Wandoujia	Specialized	HTTPS ✔
PC Online	Specialized	HTTPS ✔
LIQU	Specialized	HTTPS ✔
25PP	Specialized	HTTPS ✔
App China	Specialized	HTTP ✗
Sougou	Specialized	HTTP ✗
AnZhi	Specialized	HTTP ✗

2.4 Relevant Technical Background

To facilitate permanent, distributed, and secure app access, we propose a novel architecture based on Ethereum/Polygon and IPFS. In this subsection, we provide the necessary background information to better understand our proposed solution.

Blockchain. A blockchain is typically a public and distributed ledger. It records transactions that are immutable, verifiable, and permanent [44]. Therefore, blockchain can be utilized as a decentralized database. The trust among different nodes is guaranteed by the so-called *consensus* (e.g., Proof of Work) instead of the authority of a specific institution. Consensus is the key for all nodes on a blockchain to maintain the same ledger in a way that the authenticity could be recognized by each node in the network.

Ethereum [26] is the second largest blockchain system and the most popular smart contract platform [55, 56]. A smart contract is a contract that has been programmed in advance with a sequence of rules and regulations for self-executing. Solidity is the primary language for programming smart contracts on Ethereum. In particular, it is a Turing-complete language, which suggests that developers could achieve arbitrary functionality on smart contracts theoretically. To prevent denial-of-service attacks, users need to consume gas fees to send transactions on Ethereum. The gas fees are paid in Ethereum's native cryptocurrency called Ether (or ETH). Recently, **Polygon** [16], Ethereum's layer-2 network, has been emerging as a result of the expensive gas fee and low throughput in Ethereum.

IPFS (Interplanetary File System) [25] is a peer-to-peer file sharing system, where files are stored in a distributed way and routed using the content-addressing [29]. IPFS was proposed because the storage of large files on blockchain is inefficient and with high costs. Specifically, all nodes in a blockchain network need to endorse the entire ledger and synchronize the files stored. As a result, unnecessary redundancy will lead to a huge waste of storage, and the latency of ledger synchronicity will also be significantly increased. To address this limitation, we can store data only in the IPFS storage nodes and keep the unique and permanent IPFS address (called IPFS hash) in the blockchain.

3 THE CORE DESIGN OF AGCHAIN

In this section, we present the core design of AGChain, including its objectives, threat model, overall design, and the two key challenges we tackled.

3.1 Design Objectives and Threat Model

Design objectives. Our goal is to develop a trustworthy blockchain-based gateway that enables permanent, distributed, and secure app delegation from existing app markets. It is worth noting that our intention is not to replace these existing markets, as the apps on AGChain ultimately originate from them. Instead, our aim is to provide an additional option for users with security concerns or those who wish to back up their apps. They can utilize AGChain as a secure proxy to download apps from third-party markets or store them permanently on the blockchain. Therefore, AGChain functions more as a gateway to existing app markets rather than a standalone market. To achieve this, we have identified the following key design objectives:

- *Permanent delegation.* To achieve the permanent storage and delegation of apps on AGChain, we utilize the immutability of blockchain technology. In doing so, we make two specific choices. Firstly, instead of developing our own blockchain infrastructure like Infnote [57], we leverage existing and well-established blockchains such as Bitcoin and Ethereum. These mainstream blockchains have a large network of nodes, accumulated over time, making them robust against various attacks. Secondly, to implement our logic on the blockchain, we opt for writing a smart contract rather than creating a virtualchain as seen in [23, 37, 42]. Smart contracts are lightweight yet powerful in terms of their Turing-completeness. Note that only apps accessed through AGChain can benefit from permanent access.
- *Distributed delegation.* Due to the high storage and gas costs associated with directly storing raw files on the blockchain (as discussed in §2.4), we need an efficient and distributed file storage solution. In this paper, we utilize IPFS for its distributed and permanent nature, unlike centralized cloud services that cannot guarantee continuous availability (some are even subject to censorship, such as Google Drive and Dropbox). The basic idea behind incorporating IPFS into AGChain is to store raw app files on IPFS while keeping their corresponding IPFS indexes on the blockchain. However, we discovered an unexpected behavior of IPFS, where files are not duplicated to other IPFS nodes unless there is a request for them. To truly achieve distribution, we establish a consortium network by leveraging IPFS gateways and crowdsourced server nodes.
- *Secure delegation.* A practically desirable objective is to enable secure app download delegation for apps obtained from markets that do not support HTTPS downloading (referred to as *unprotected markets* hereafter). This objective has significant implications for millions of Chinese market users (as discussed in §2.3). Specifically, we aim to securely retrieve apps from existing app markets without relying on a dedicated and trusted network path. In addition to ensuring download security, we also need to prevent repackaged apps [59] being uploaded to AGChain.

Besides the core objectives, we also aim for *sustainable delegation* to ensure AGChain's self-sustainability by providing monetary incentives to crowdsourced server nodes. The implementation details of secure delegation and sustainable delegation will be explained in §4. In this section, our focus is on achieving permanent and distributed delegation for AGChain.

Threat Model. In this section, we formally define the threat model by specifying the adversary's capabilities and the system's security assumptions. The formalization is structured as follows:

Notation and Definitions.

- Let $\mathcal{U}$ denote the set of all potential adversaries.

- Let $\mathcal{S} = \{F, S, C, I\}$ represent the system components (see Fig. 1; to be introduced in §3.2):
  - F: Front-end
  - S: Server Nodes
  - C: Smart Contract
  - I: IPFS Network
- Define C_A as the set of capabilities of an adversary $A \in \mathcal{U}$.
- Define $\mathcal{A}$ as the set of assumptions about the system's security.

Adversary Capabilities. For any adversary $A \in \mathcal{U}$, the capabilities C_A are defined as:

$$C_A = \{\text{Intercept}(F \leftrightarrow S), \text{UploadRepackagedApp}, \text{Compromise}(F), \text{Access}(C), \text{Access}(I)\}$$

Where:

- $\text{Intercept}(F \leftrightarrow S)$: Ability to intercept unprotected network traffic between the front-end F and server nodes S.
- UploadRepackagedApp: Capability to upload repackaged or malicious applications to legitimate app marketplaces.
- $\text{Compromise}(F)$: Potential to compromise or replace the front-end code F since it operates on the user side.
- $\text{Access}(C)$: Ability to access and interact with the smart contract C.
- $\text{Access}(I)$: Ability to access and interact with the IPFS network I.

Security Assumptions. The following assumptions $\mathcal{A}$ constrain the adversary's capabilities to ensure system security:

$$\mathcal{A} = \{\neg\text{BreakTLS}, \neg\text{BreakHash}, \neg\text{CompromiseChain}, \neg\text{CompromiseIPFS}, \neg\text{Exploit}(C), \neg\text{Exploit}(S)\}$$

Where:

- ¬BreakTLS: Adversary A cannot break the Transport Layer Security (TLS) protocol.
- ¬BreakHash: Adversary A cannot break cryptographic hash functions such as SHA-256.
- ¬CompromiseChain: Adversary A cannot compromise the underlying blockchain infrastructures.
- ¬CompromiseIPFS: Adversary A cannot breach the security of the IPFS storage mechanisms.
- $\neg\text{Exploit}(C)$, $\neg\text{Exploit}(S)$: Adversary A cannot exploit the smart contract C, nor the server nodes S because we can leverage the recent advances on verification of smart contracts [43] and write SGX-enabled[3] server code [46] to enhance our contract and server code's security, although this is out of the scope of this paper.

3.2 The Overall System Design

Architecture. Fig. 1 presents AGChain's high-level design. As highlighted in the green color, it has four components as follows:

- *Smart contract.* The most important component is a novel (gas-efficient) smart contract, which stores all app metadata on chain and duplicates them on most of the Ethereum nodes worldwide. With these data (including IPFS file indexes) and their programmed storing and retrieving logic, this smart contract is the actual control party of AGChain's entire logic.

- *IPFS network.* Another core component is an IPFS (consortium) network, which stores raw app files in a distributed manner. The stored apps then can be automatically routed and retrieved through IPFS's content-addressing [17]. Note that with the smart contract and IPFS components, we guarantee the permanent and decentralized app access in AGChain.

[3]SGX (Software Guard eXtension) is a TEE (trusted execution environment) technique developed by Intel to protect selected code and data.

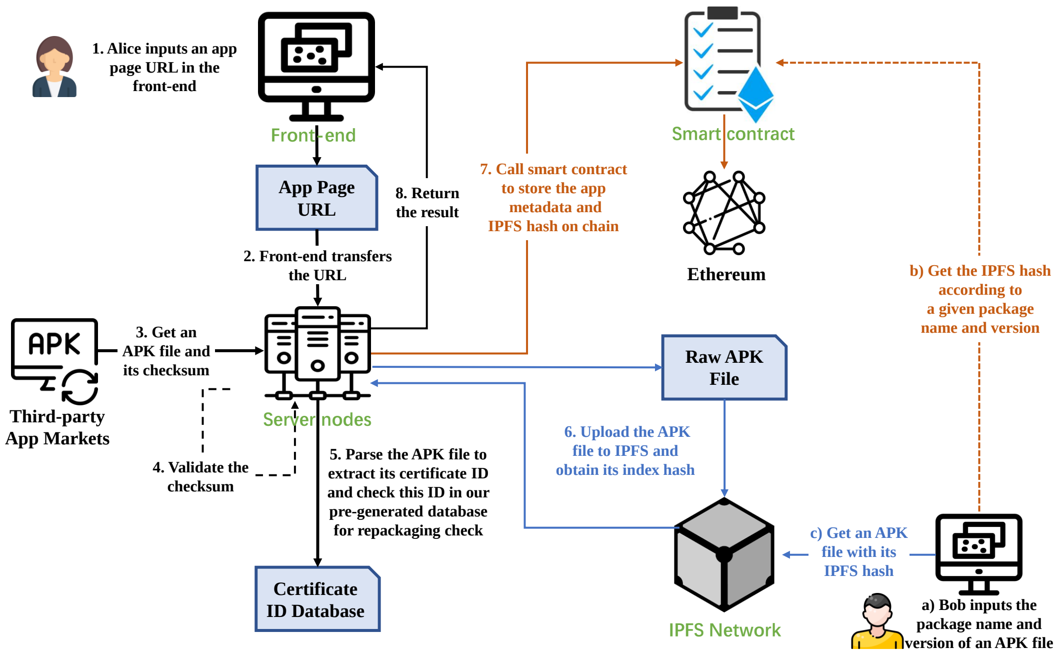

1. Alice inputs an app page URL in the front-end
Front-end
App Page URL
2. Front-end transfers the URL
8. Return the result
7. Call smart contract to store the app metadata and IPFS hash on chain
Smart contract
Ethereum
b) Get the IPFS hash according to a given package name and version
APK
Third-party App Markets
3. Get an APK file and its checksum
Server nodes
4. Validate the checksum
5. Parse the APK file to extract its certificate ID and check this ID in our pre-generated database for repackaging check
Certificate ID Database
Raw APK File
6. Upload the APK file to IPFS and obtain its index hash
IPFS Network
c) Get an APK file with its IPFS hash
a) Bob inputs the package name and version of an APK file

Fig. 1. A high-level workflow of AGChain, which consists of four components marked in the green color.

- *Server nodes.* To achieve app download security and repackaging checking, we also need server(s) to retrieve apps from existing markets, inspect their security, and schedule their uploading as these tasks cannot be performed in blockchain. Since any machine with our server code could be a server node, we propose an incentive mechanism (details in §4.3) to motivate servers to join AGChain. These *crowdsourced* server nodes further enhance the decentralization.

- *Front-end.* Finally, we provide a front-end web interface to help uploaders and downloaders interact with AGChain. Note that for app downloading, our front-end directly communicates with smart contract and IPFS without the server.

Upload workflow. Fig. 1 also shows the overall workflow of AGChain. As shown in the left part, Alice wants to securely download an app (e.g., Alice is a security-sensitive user) or permanently store an app on chain for future usage (e.g., Alice is a developer or a user who wants to backup the current version of an app). She then acts as an app uploader:

1. Alice just needs to input the original app page URL and clicks the "upload" button in the front-end. AGChain automatically finishes all the remaining steps.
2. The front-end transfers the URL to one server node.
3. After knowing the URL, the server analyzes the corresponding app page to obtain the download URL that points to the actual APK file (the file format used by Android apps), and retrieves it from its original market.
4. For the app markets using insecure app downloading (e.g., those seven markets in Table 1), AGChain performs one more step to check whether the retrieved file has been tampered with during network transmission.

5. For all third-party markets, we conduct repackaging checks to prevent repackaged apps [59] from polluting AGChain. We propose a lightweight yet effective certificate ID-based mechanism; see details in §4.2. During this process, we also parse the APK file to extract its package name and version (besides the certificate ID).
6. The server then uploads the raw APK file to IPFS and obtains its corresponding IPFS hash (i.e., the file index).
7. Finally, the server invokes the smart contract to store all app metadata and IPFS hash on chain. We choose the widely-used Ethereum and its layer-2 network Polygon (see §2.4) as our underlying blockchain.
8. To avoid Alice from waiting a long time for blockchain transaction confirmation (typically 9∼13 seconds, according to our tests), AGChain simultaneously returns the result of package name, app version, and IPFS hash.

Download workflow. As shown in the right part of Fig. 1, Bob acts as an app downloader to download apps (e.g., the app uploaded by Alice) that are already in AGChain:

a) Bob inputs (or browses) the package name and version of the app that he wants to download in the front-end.
b) The front-end then automatically invokes the smart contract to retrieve the corresponding IPFS hash.
c) The front-end further locates the nearest IPFS network node to download the APK file and returns it to Bob.

Major challenges. In the course of developing AGChain, we identify two previously neglected challenges:

C1: *How to minimize gas costs in the smart contract?* As each app upload requires a blockchain transaction and incurs gas fees, it is crucial to minimize these costs in our smart contract. Although there have been some code-level gas optimizations [22, 28, 30, 48], they are still insufficient. In §3.3, we introduce design-level optimizations that significantly reduce gas costs by a factor of 16.

C2: *How to enable fully distributed IPFS storage?* As discussed in §3.1, we discovered that IPFS files are only stored in the original IPFS node and cached at IPFS gateways upon request. If the original node goes offline, the file becomes inaccessible to the entire IPFS network (availability is restored when another node adds the same content). To achieve distributed IPFS storage, we establish an IPFS consortium network (§3.4) that proactively requests IPFS gateways and crowdsourced server nodes to back up files.

3.3 Gas-Efficient Smart Contract for App Metadata Storing and Retrieving

The smart contract is a core component of AGChain and also serves as a control entity. On one hand, the server invokes this contract to store app metadata on-chain (see step 7 in Fig. 1). On the other hand, the front-end relies on this contract to retrieve the IPFS hash (see step b in Fig. 1). The most important contract function is `storeApp()`. By sending Ethereum transactions to invoke this function, we not only store app metadata on-chain but also replicate this data across most Ethereum nodes worldwide.

In this subsection, we introduce a series of mechanisms aimed at reducing gas costs for app metadata storage (i.e., app uploads), eliminating gas costs for metadata retrieval (i.e., app downloads), and implementing a whitelist mechanism to prevent misuse of our contract functions. These design-level optimizations are significantly more efficient than code-level optimizations [22, 28, 30, 48] and provide valuable guidance for future smart contract designs.

Minimizing gas costs via log-based contract storage. We found that a major source of gas inefficiency comes from data storage in the smart contract. Like many other smart contracts, AGChain needs to store app metadata as a structure defined in the contract. However, such data storage operation, via the `SSTORE` instruction in Ethereum Virtual Machine (EVM), changes the internal block states and Ethereum's world state [3]. Therefore, it is expensive, costing 20,000 Gas[4] per operation [50]. Since numerous app metadata records will be stored in AGChain, it would cost a large amount of gas fees if we use the traditional contract structure. Fortunately, we identify a logging interface in the EVM, which can be used to permanently log data in transaction receipts via the `LOG` instruction [50]. Since it changes only the block headers and does not need to change the world state, only 375 Gas is consumed per operation. The underlying logging mechanism is complicated, and we refer interested readers to the Ethereum yellow paper [1, 50] for more details. While log-based contract storage dramatically reduces gas fees, it has no structure information. We thus ask our server nodes to recover the app metadata structure, which includes the app package name, app package version, app certificate serial number, the original market page URL, the repackaging detection result, and the important IPFS hash.

Offloading on-chain duplicate check to server nodes. Another major source of gas costs originates from the duplicate check, which checks duplicated app metadata before storing it on chain. Originally, we deployed such a check in the smart contract, but we found that it is costly since each check needs to iterate over the entire app metadata structure. Moreover, an in-contract structure has to be maintained, which causes the first log-based optimization unadoptable. Therefore, we offload this on-chain duplicate check to server nodes, which query the latest app metadata from the smart contract before uploading any records. If a duplicate exists, the uploading will stop.

Further reducing gas costs via batch uploads. With the above two default optimization mechanisms, we significantly reduce gas costs by a factor of 15.75 (0.001347 v.s. 0.0000855 Ether, before and after the optimization). We further reduce the costs by providing a back-end interface of batch uploads. Our experiment shows that by batching 10 uploads together, we save gas by a factor of 2.02 (compared with 10 times of normal uploads). This factor increases to 2.65 for a batch upload of 100 records. These suggest that for a large number of app uploads (e.g., a company uploads all its apps), we can use batch uploads to minimize gas costs.

Eliminating gas costs for all app downloads. While app uploading certainly consumes gas, we find a way to eliminate gas costs for all app downloads. For a normal contract data structure, we originally used the `view` function modifier to describe the data retrieving contract function since it does not change any contract state. Invoking such a view-only function (even by external parties) will not initiate any blockchain transaction, and thus no gas fee is needed. However, since we have switched to log-based contract storage, we create a bloom filter [1, 5] to quickly locate the block headers containing our data logs, regenerate the original logs, and extract IPFS hashes from them. Since this task is performed only at the server side via the web3 Python APIs, the contract side will not cost any gas.

Comparative analysis with traditional methods. To summarize the efficiency gains of our proposed mechanisms, we compare them against traditional smart contract storage and operation methods in Table 2. These comparisons highlight the substantial gas savings achieved through our optimized smart contract design. By leveraging EVM features and offloading certain operations off-chain, we provide a more cost-effective and scalable solution for app metadata management on the blockchain. However, offloading operations like duplicate checks and data retrieval to server nodes requires stringent security measures to prevent unauthorized access or manipulation, which we assume is handled by the SGX-enabled server code in our threat model; see §3.1.

[4]The gas cost/fee is the product of gas price (or Gwei) and the Gas consumed.

Table 2. Comparative Gas Costs of Traditional and AGChain's Optimized Methods.

Mechanism	**Traditional Gas Cost**	**AGChain's Gas Cost**	**Efficiency Gain**
Log-Based Storage	20,000 Gas per `SSTORE`	375 Gas per `LOG` + 895 additional Gas	~15.75× reduction
Batch Uploads (10 records)	200,000 Gas (10 × 20,000)	6,286 Gas	~31.8× reduction
Batch Uploads (100 records)	2,000,000 Gas (100 × 20,000)	47,918 Gas	~41.7× reduction
Duplicate Checks (On-Chain)	Variable (high)	Off-Chain (negligible)	Significant reduction
Data Retrieval (`view` Functions)	0 Gas externally, variable in contract call	0 Gas via off-chain logs	Complete elimination

Achieving a whitelist mechanism for access control. Since smart contract has no access control mechanism, a contract function can be invoked by anyone. This implies that an adversary can invoke our `storeApp()` function to upload any app in our scenario. To present malicious uploads from interfering AGChain's data, we implement a lightweight whitelist mechanism that consists of two function modifiers, `onlyOwner(address caller)` and `onlyWhitelist(address caller)`, where the parameter `address` is the invoking party's Ethereum account address. By enforcing the `onlyWhitelist (msg.sender)` modifier to check the caller of `storeApp()`, we can guarantee that only an account in the whitelist can upload apps. We also implement two contract functions to add or delete an account address from the whitelist, and they are enforced by the `onlyOwner(msg.sender)` modifier. Note that the owner is our contract creator, and we gradually add each authorized server node into the whitelist. By designing such a hierarchical whitelist, we can achieve effective access control to avoid malicious data injecting into AGChain.

3.4 IPFS Consortium Network for Distributed File Uploading and Downloading

To overcome challenge C2, AGChain utilizes IPFS gateways and crowdsourced server nodes to cache or backup apps, ensuring their true distribution within the IPFS network. By establishing these gateways and servers, AGChain effectively creates an IPFS consortium network, enabling distributed app access.

Periodically caching app files at IPFS gateways. As described in challenge C2, an IPFS file is only accessible through the original IPFS node and is distributed via content-addressing [17]. However, if the original node becomes offline, the file becomes inaccessible to the entire IPFS network. Fortunately, our experiment revealed that an IPFS gateway caches files for a certain duration when accessed through the gateway. Leveraging this observation, we intentionally simulate user requests to cache apps at IPFS gateways. To achieve this, we deploy a script on the server that sends periodic file requests through various IPFS gateways. These requests are sent before the IPFS garbage collector cleans our app files, ensuring that copies of the app files are always available in IPFS gateways.

Timely backing up apps at crowdsourced server nodes. To achieve fully distributed app storage, we utilize each crowdsourced server as an IPFS storage node and ensure timely backups of apps in our IPFS consortium network. Each server node initializes by running the `ipfs daemon` command to function as an IPFS node. It then executes a consortium synchronization script, which obtains a list of IPFS hashes for the apps in AGChain and retrieves the corresponding raw app files from the IPFS network. To prevent the IPFS garbage collection from removing these files, the script locally pins the raw apps using the `ipfs pin` command. This approach enhances data redundancy in AGChain and enhances the distribution of app storage.

Identifying fast IPFS gateways for app downloading. In addition to distributed app uploads, we introduce a mechanism to enable distributed and fast app downloading in AGChain. The front-end of AGChain performs RTT (Round-Trip Time) tests on public IPFS gateways and selects the gateway with the lowest RTT for downloading the raw app file. This approach enhances the performance of app downloading in IPFS while also mitigating the risk of potential censorship by certain IPFS gateways.

Based on the above design, we clarify how the proposed IPFS consortium network maintains its operational logistics, maintenance, scalability, and performance as follows.

Operational Logistics and Maintenance of the IPFS Consortium Network: The IPFS consortium network utilizes a combination of IPFS gateways and crowdsourced server nodes for file distribution. Each crowdsourced server acts as an independent IPFS node, running the `ipfs daemon` to maintain a connection to the IPFS network. Periodic synchronization of app files is carried out by a dedicated script, which retrieves the latest IPFS hashes and pins these files locally using the `ipfs pin` command. This ensures that each app file is replicated across multiple nodes, increasing redundancy and mitigating the risk of file loss if a node becomes unavailable. Additionally, the script actively monitors the availability of files to minimize maintenance overhead and prevent files from being removed by the IPFS garbage collector.

Scalability of the Consortium Network: The scalability of the IPFS consortium is achieved through the use of both IPFS gateways and crowdsourced nodes that provide caching and file backup. The system can easily scale by adding more crowdsourced nodes or leveraging additional IPFS gateways to handle increased load. This distributed approach ensures that file availability and redundancy grow alongside the number of contributing nodes. The system can adapt to the varying storage demands of the network by dynamically pinning files across different locations, making it highly flexible and scalable as more nodes join the consortium.

Performance and Availability Across Varying Loads: To ensure consistent performance, AGChain periodically tests the RTT of available IPFS gateways and dynamically selects the gateway with the lowest RTT to improve download speeds. By utilizing geographically distributed gateways, AGChain can handle regional variations in access and mitigate network congestion. The combination of periodic caching at multiple gateways and distributed backups at crowdsourced nodes allows the system to maintain high availability and fast file access across varying loads and geographic distributions. Moreover, the use of multiple nodes helps in load balancing, ensuring that no single gateway is overwhelmed by requests, thereby enhancing overall network resilience.

4 MAKING IT SECURE AND SUSTAINABLE

So far, we have designed AGChain to be permanent and distributed through contract-based distribution and IPFS-based storage. However, to make it secure and sustainable, we still need to address three AGChain-specific system issues:

C3: *How to securely retrieve apps from the app markets without a network security guarantee?* Recall that our server needs to retrieve apps from existing markets before uploading them to IPFS. This process is relatively simple for markets that use HTTPS, but it becomes challenging for those that rely on insecure HTTP (as discussed in §2.3) since there is no inherent network security guarantee. To tackle this issue, we propose two modes of secure app retrieval in §4.1.

C4: *How to avoid repackaged apps from polluting our market?* By addressing challenge C3, we ensure that the retrieved app is identical to the one available in its original app market. However, in the case of a third-party app market, there is a possibility that the app has already been repackaged before our secure retrieval. Consequently, we require a mechanism, as proposed in §4.2, to detect repackaged apps [59] that may differ from their official versions on Google Play.

C5: *How to make AGChain self-sustainable?* As stated in §3.2, each server node in the crowdsourced network consumes computer resources and incurs gas fees for app uploading in AGChain. This becomes unsustainable without a monetary mechanism to incentivize these nodes. Considering that uploaders benefit from AGChain by advertising their apps or securely storing them for security research purposes, it is reasonable to impose an upload fee on them to compensate the crowdsourced servers. We will introduce this mechanism in §4.3. By implementing upload fees, we can also deter spammers from abusing AGChain.

4.1 Secure App Retrieval from (Existing) Unprotected Markets

In this subsection, we present two methods that collectively achieve secure app retrieval even for those unprotected markets, e.g., seven markets using HTTP downloading in Table 1.

Secure app downloading via checksums. We find that although those seven markets use HTTP to download apps, most of them allow users to browse app pages via HTTPS (e.g., https://zhushou.360.cn/detail/index/soft_id/95487). Additionally, we can extract app APK file checksums (e.g., MD5) from the HTML source code of these pages. For instance, three app markets (Baidu, 360, and AppChina) directly embed the checksums in their app download URLs. In the case of Lenovo MM, the checksum is embedded in the `<script>` data section of the app's HTML page. By analyzing the HTML pages of these markets, we can securely obtain app checksums and compare them with the calculated checksums of the app APK files we retrieve via HTTP. If they match, we conclude that the retrieved app has not been tampered with by adversaries. Furthermore, although Sogou market does not provide checksum information, we discover that its HTTP download URL can be converted into an HTTPS version. Consequently, we can ensure guaranteed app download security for five out of the total seven unprotected markets.

Alternative security with no checksums. To address the lack of checksums in the remaining app markets such as Meizu and Anzhi, we propose an alternative mechanism to ensure download security. The fundamental concept is to cross-check each downloaded app with its Google Play counterpart. This can be achieved by utilizing the AndroZoo app repository [24], which houses a vast collection of over ten million apps sourced from Google Play. By comparing a downloaded app with the apps in the AndroZoo repository, we can determine whether it is present. If it is, we conclude that the downloaded app from a third-party market has not been compromised during the download process. Additionally, even if an app is not found in the repository, we can extract its developer certificate and verify if it belongs to a Google Play developer. Given the dynamic nature of the AndroZoo repository, which continuously updates to reflect changes in Google Play, we have a high level of confidence in covering most apps through either app-level or developer-level checking. For the rare cases where some apps are not adequately covered, AGChain will issue a warning indicating that these apps might not have been securely retrieved.

4.2 Exploiting App Certificate Info for Repackaging Detection

In this subsection, we present a mechanism of exploiting app certificate to *accurately* detect repackaged apps that might be uploaded to AGChain from third-party app markets.

In contrast to traditional app repackaging detection methods [34, 36, 59], the identifier of apps uploaded to AGChain, namely the app package name, remains fixed. This fixed package name identifier allows us to locate the corresponding official app on Google Play. The challenge lies in differentiating between these two versions of apps and specifically determining whether they originate from the same developer. To address this challenge, we examine the data structure of the app certificate, which includes essential fields such as the certificate serial number, issuer, subject, and X509v3 extensions. Among these fields, we identify the serial number (e.g., 0x706a633e) as the most lightweight metric that adversaries cannot manipulate since they lack access to the developers' app signing key. On the other hand, fields like issuer and subject are susceptible to manipulation, and X509v3 is more complex compared to the serial number.

We further experimentally validate our detection idea by utilizing a dataset [34] consisting of 15,297 *pairs* of repackaged Android apps. Each pair consists of an original app and its corresponding repackaged version. This dataset serves as our ground truth. To facilitate our analysis, we developed a script that automatically extracts the package name and serial number from any given app APK file. This script utilizes the widely used Androguard library [14] for parsing APK files. It is worth noting that we have integrated this script into AGChain's server code. By running our script on the 15,297 app pairs, we discover a total of 2,270 pairs that share the same package name, while none of the pairs have the same serial number. Moreover, for the remaining 13,027 repackaged pairs, each pair has distinct package names. This experiment provides compelling evidence that our serial number-based mechanism achieves 100% accuracy in detecting repackaged apps.

4.3 Charging Upload Fees to Maintain the Platform Self-Sustainability

In this subsection, we design a mechanism of charging app upload fees to pay crowdsourced server nodes in AGChain so that the entire AGChain platform is sustainable.

We thus revise the original design of AGChain to charge upload fees just before each server node invokes (IPFS and) smart contract. More specifically, we add two more steps between step 5 and 6 in Fig. 1. The first step is to send an estimated transaction fee to our smart contract, for which we add one more function called `DonateGasFee()` in our smart contract. We set this function `payable` so that the user side (in the form of our JavaScript code) can invoke it with a fee, e.g., `DonateGasFee().send(from: userAccount, value: fee)`. After this call is successfully executed in blockchain, the user side will receive a transaction ID. In the second step, our JavaScript code automatically sends this transaction ID (and a few other metadata) to the server for verification.

We now explain how our JavaScript code estimates the gas fee and how the server verifies the payment transaction. To calculate a gas fee at the user side, we invoke a web3 JavaScript function called `estimateGas()` to estimate the required gas of executing the transaction in the EVM. For the transaction verification at the server, we first query the transaction according to its ID, and then determine (i) whether the destination address of this transaction is our smart contract, (ii) whether the upload fee exceeds our estimated gas, and (iii) whether this transaction is never used before. Only when the three conditions are all satisfied, the server then continues the actual app uploading to IPFS and Ethereum/Polygon.

5 IMPLEMENTATION

We have implemented a prototype of AGChain, utilizing Ethereum smart contract and IPFS. Fig. 2 shows a screenshot of the front-end homepage. In this section, we summarize the implementation details of AGChain. The current prototype consists of 2,485 LOC (lines of code), excluding all the library code used. Table 3 lists a breakdown of LOC across different components and programming languages.

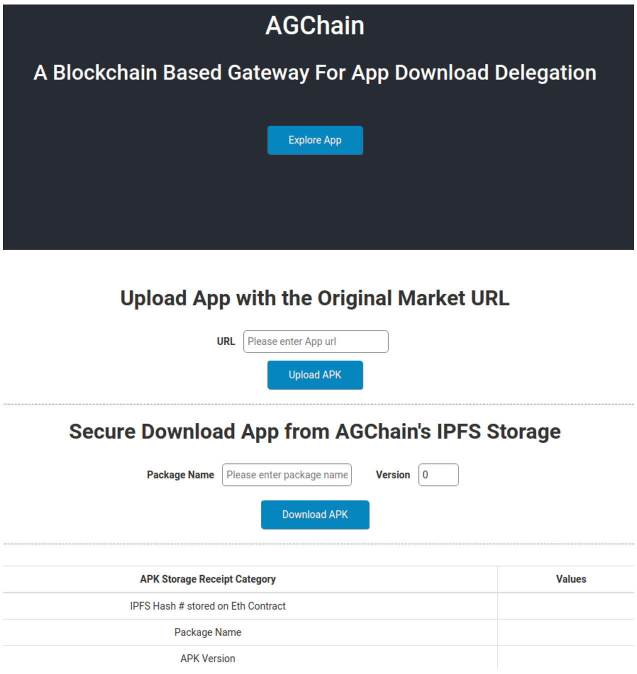
Front-end. We implement the front-end user interface using the React web framework. Hence, the HTML and CSS code is minimal, and some HTML contents are also dynamically generated using JavaScript. Besides user interfaces, we write 302 JavaScript LOC on top of the web3.js library [12] to query our smart contract for retrieving IPFS hashes. To execute IPFS commands in JavaScript for app downloading, we write additional 80 JavaScript LOC based on the js-ipfs library [11]. Overall, the front-end implementation consists of approximately 862 lines of code.

Server node. We implement our server code in a total of 1,572 LOC and run it on the AWS (Amazon Web Services). Specifically, we write 849 Java LOC to handle requests from the front-end, securely download apps from existing markets, perform repackaging checks, and upload APK

Table 3. A breakdown of LOC (lines of code) in AGChain.

	Front-end (F)	Server (S)	Contract (C)	IPFS (I)	AGChain
JavaScript	781		(302 in F)*	(80 in F)	781
Java		849		(36 in S)	849
Python		723	(445 in S)		723
Solidity			51		51
CSS	81				81
Sum	862	1,572	51 + (747)	(116)	2,485

*This means that 302 lines of code in front-end are related to smart contract. Other brackets, such as (445 in S) and (80 in F), are similar.



AGChain

A Blockchain Based Gateway For App Download Delegation

Explore App

Upload App with the Original Market URL

URL Please enter App url

Upload APK

Secure Download App from AGChain's IPFS Storage

Package Name Please enter package name Version 0

Download APK

APK Storage Receipt Category	Values
IPFS Hash # stored on Eth Contract	
Package Name	
APK Version	

Fig. 2. A screenshot of AGChain's front-end homepage (cutted). Besides uploading and downloading apps, users can click the "Explore App" button to browse the apps stored in AGChain. The bottom part will output the metadata of each new upload, which can help users immediately download the app from AGChain using the "Download APK" button.

files to IPFS. Moreover, we write 445 Python LOC to leverage web3 Python APIs [13] to interact with smart contract. Lastly, for APK file parsing (used in repackaging checks), we leverage the Androguard library [14] and write 278 Python LOC on top of it.

One of the main tasks of the server node is to retrieve apps from the market URLs provided by users. However, these URLs often only consist of page URLs, such as https://shouji.baidu.com/software/11569169.html, instead of direct download URLs. In order to automatically extract the download URLs, we utilize the understanding that each market follows a distinct pattern when transitioning from a page URL to a download URL. As a result, we conduct pre-analysis of these markets to obtain their respective download URL patterns. For the majority of markets, their download URLs can be directly extracted from the HTML tags, similar to the aforementioned Baidu market example. However, a few markets, such as Meizu and AnZhi, require the calculation of URLs from their JavaScript code. Currently, our prototype has analyzed the download patterns of all seven markets that require AGChain's secure app download delegation.

Smart contract. We implement the smart contract with 51 LOC in the Solidity language, which was reduced from 128 LOC in the earlier version since we no longer define in-contract data structures (see §3.3). Therefore, our smart contract mainly describes a list of functions, e.g., `storeApp()` for uploading app metadata to the blockchain. In particular, to avoid the smart contract being misused by unauthorized parties, we set that only whitelisted server nodes can invoke the `storeApp()` function by adding a function modifier to check the transaction sender. However, this also prevents the front-end's `estimateGas()` web3 API from estimating gas (see §4.3). To address this issue, we create an additional function called `storeApp_estimate()` that duplicates `storeApp()`'s functionality but does not execute data push operations. This function will be executed by the EVM instead of AGChain transactions.

IPFS module. As we do not make any modifications to the IPFS network, the implementation of IPFS-related code is carried out in other components. For instance, we have developed 80 JavaScript LOC for the front-end to facilitate the downloading of apps from IPFS. Similarly, the server node is responsible for uploading apps to IPFS, and we have implemented this functionality using 36 lines of Java code (excluding file operation code). To activate the IPFS node on the server, we simply run the `ipfs daemon` command.

6 EVALUATION

In this section, we first experimentally evaluate the performance and gas costs of AGChain, then empirically demonstrate its security effectiveness and decentralization. We have deployed AGChain to the Ethereum blockchain (tested in the Ethereum Rinkeby environment) in 2021 and further deployed it to Polygon (a widely-used Ethereum layer-2 network; see §2.4) in 2022 for a real-world use.

6.1 Performance

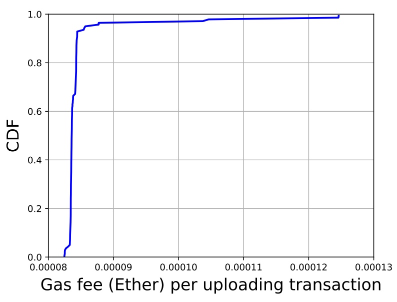
To fairly evaluate the additional time introduced by AGChain, we use our server code to record both normal downloading time (part of step 3 in Fig. 1) and AGChain's processing time (the rest of step 3 and steps 4 to 6). Note that we do not count step 8 as part of AGChain's overhead, because we simultaneously return results to users without waiting for the transaction to be confirmed. Totally, we conduct 200 app tests from seven markets (one market, Sogou, is no longer available in 2022 when we test in Polygon) that require secure delegation. Moreover, we perform these tests in different days to minimize the impact of different network conditions.

Table 4 lists the average results of each tested market for the Ethereum Rinkeby environment and the layer-2 Polygon network. Note that the normal downloading time simply relies on the network quality between app markets and our AWS server instead of APK file sizes. On top of the normal downloading time, AGChain introduces three steps of additional processing times, including (i) about one second (or 1s) for extracting and validating checksums; (ii) ~0.4s for performing repackaging checks; and (iii) ~0.4s for uploading apps to IPFS. With these, the overall overhead introduced by AGChain is from 2.16% to 20.56%, with the median and average of 12.05% and 11.39%, respectively. We thus conclude that AGChain's performance overhead is around 12%, a reasonable performance cost for a blockchain-based system.

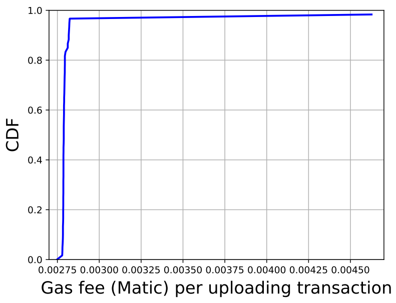
Table 4. Average processing time introduced by AGChain.

Market ID	Market Name	APK Size (Mb)	Normal Download Time (ms)	Additional Processing Time (ms)							
				Checksum		Repackaging		IPFS Upload		Overall %	
				E*	P*	E	P	E	P	E	P
1	Baidu Market	20.09	71063.2	820.1	1324.9	377.1	557.8	340.7	599.5	2.16%	3.49%
2	360 Market	27.26	10105.5	917.0	1333.8	433.8	393.0	429.1	351.2	17.61%	20.56%
3	Lenovo MM	20.87	14819.2	1203.3	1245.3	408.6	391.9	415.8	277.5	13.68%	12.92%
4	APP China	20.63	20090.9	1328.0	1263.6	355.8	380.5	387.4	271.7	10.31%	9.54%
5	Sougou	30.52	13854.8	990.1	-	444.9	-	393.2	-	13.20%	-
6	Meizu Market	25.29	21875.8	1870.7	1406.2	369.8	398.1	396.1	330.1	12.05%	9.76%
7	AnZhi	23.86	17345.3	1013.0	1596.1	414.3	398.5	432.7	352.4	10.72%	13.53%

*: E and P represent the Ethereum Rinkeby environment and the layer-2 Polygon network, respectively.
-: this market is no longer available in 2022.



(a) In Ethereum (1 Ether≈2,300 USD)



(b) In Polygon (1 Matic≈1 USD)

Fig. 3. CDF plots of gas fees per app uploading in AGChain.

6.2 Gas Costs

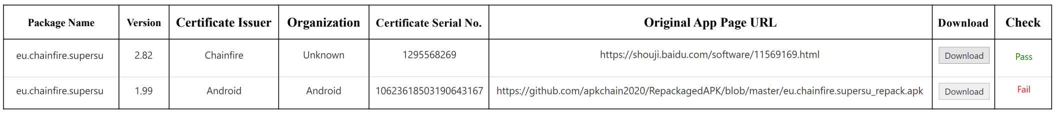
Throughout the 200 performance tests, we also collected the corresponding gas fees in Ether (for the Ethereum Rinkeby environment) and in Matic (for the layer-2 Polygon network). As of 31 December 2023, the value of one Ether is approximately 2,300 USD, while one Matic is around 1 USD.

Fig. 3 shows the CDF (cumulative distribution function) plot per app uploading in AGChain, for both Ethereum and Polygon. We can see that for the Ethereum Rinkeby environment, over 95% gas fees are in the range of 0.00008245 Ether (0.1896 USD) and 0.00008773 Ether (0.2018 USD). Only four tests consumed a gas fee over 0.0001 Ether, ranging from 0.00010374 and 0.00012461 Ether. The average of all the gas fees of AGChain in the Ethereum Rinkeby environment is 0.00008466 Ether (0.1947 USD). In contrast, the gas fees of AGChain in the Polygon network is even much smaller, with an average of only 0.002821 Matic (0.0028 USD). Since only uploads in AGChain cost gas and one upload can serve all future downloads, we believe that such gas costs are quite acceptable for real-world deployment.

6.3 Security

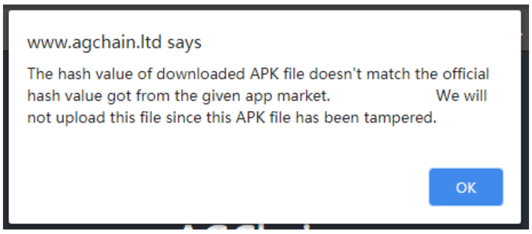
Since there are no real-world attacks against AGChain, we mimic a MITM (Man-In-The-Middle) attack and a repackaging attack to demonstrate AGChain's security effectiveness.

Preventing a MITM attack. To mimic the MITM attack, we originally tried to control the network traffic of our AWS server using the mitmproxy tool [15] (since we cannot redirect app markets' traffic as real adversaries). However, it turned out that AWS disallows this. Hence, we



Package Name	Version	Certificate Issuer	Organization	Certificate Serial No.	Original App Page URL	Download	Check
eu.chainfire.supersu	2.82	Chainfire	Unknown	1295568269	https://shouji.baidu.com/software/11569169.html	Download	Pass
eu.chainfire.supersu	1.99	Android	Android	10623618503190643167	https://github.com/apkchain2020/RepackagedAPK/blob/master/eu.chainfire.supersu_repack.apk	Download	Fail

Fig. 4. A screenshot (with the surrounding table) to demonstrate a repackaged app successfully detected by AGChain.



www.agchain.ltd says

The hash value of downloaded APK file doesn't match the official hash value got from the given app market. We will not upload this file since this APK file has been tampered.

Fig. 5. A screenshot of AGChain defending against a MITM attack.

have to redirect the target app page URL directly in our server code. Specifically, when a user inputs this particular Tudou Video URL (http://www.appchina.com/app/com.tudou.android) in AGChain, the actual app download URL will become the URL of a similar yet fake app (https://github.com/apkchain2020/RepackagedAPK/blob/master/com.tudouship.android_4159.apk) we prepared. During this process, AGChain finds that the MD5 retrieved from the original app page is f5580d6a58bb9d97c27929f1a9c585f1 while the MD5 calculated from the downloaded APK file is a05e5187f4e9eb434bc3bbd792e35c54. Since these two checksums are different, AGChain shows an alert like Fig. 5 to the user, and does not allow this app to be uploaded.

Detecting a repackaged app. To mimic the repackaging attack, we first select a target repackaged app that has ground truth as in [34] and also appears in Chinese app markets. Our choice is the SuperSu rooting app on the Baidu market (https://shouji.baidu.com/software/11569169.html). We then upload this original app to AGChain and find that it "passes" the repackaging check, as shown in Fig. 4. We further upload the repackaged app via the URL of https://github.com/apkchain2020/RepackagedAPK/blob/master/eu.chainfire.supersu_repack.apk. This time AGChain finds that the certificate serial number is different from that in our pre-generated certificate ID database using apps. Indeed, Fig. 4 also shows that the two serial numbers are different. Hence, for the repackaged app, it "fails" the check.

6.4 Decentralization

During the 200 performance tests in different timings, we also identify IPFS gateways in over 20 different locations worldwide, which demonstrate the decentralization of AGChain.

Table 5 provides a comprehensive list of the identified IPFS gateways. We can see that these gateways are distributed across 21 different locations around the globe. Around a half, ten, gateways are located in the United States (US). This is probably because many IPFS nodes run in cloud servers, which are mainly provided by the US companies. Other than the US, Europe holds seven gateways out of the remaining 11. Compared with the US and Europe, there are only three gateways in Asia (with the remaining gateway in Canada). However, we anticipate that as FinTech gains popularity in Asia [35], more IPFS nodes will be deployed locally, resulting in faster connections. Additionally, according to the RTT result in Table 5, nearby IPFS gateways usually have shorter RTTs, which

Table 5. The IPFS gateways identified in 21 different locations.

Gateway Domain	IP Address	Location	RTT (s)
ipfs.jbb.one	47.52.139.252	Hong Kong SAR	0.04
ipfs.smartsignature.io	13.231.230.12	Tokyo, Japan	0.07
10.via0.com	104.27.129.45	San Francisco, U.S.A	0.13
ipfs.kavin.rocks	104.28.5.229	Dallas, U.S.A	0.14
ipfs.runfission.com	34.233.130.24	Ashburn, U.S.A	0.25
ipfs.k1ic.com	39.101.143.85	Beijing, China	0.51
ipfs.2read.net	195.201.149.81	Gunzenhausen, DE	0.55
ipfs.drink.cafe	98.126.159.6	Orange, U.S.A	0.56
gateway.pinata.cloud	165.227.144.202	Frankfurt, Germany	0.56
ipfs.telos.miami	138.68.29.104	Santa Clara, U.S.A	0.57
hardbin.com	174.138.8.194	Amsterdam, NL	0.57
ipfs.fleek.co	44.240.5.243	Portland, U.S.A	0.57
ipfs.greyh.at	35.208.63.54	Council Bluffs, U.S.A	0.69
gateway.temporal.cloud	207.6.222.55	Surrey, Canada	0.70
ipfs.azurewebsites.net	13.66.138.105	Redmond, U.S.A	0.72
ipfs.best-practice.se	193.11.118.5	Eskilstuna, Sweden	0.73
ipfs.overpi.com	66.228.43.184	Cedar Knolls, U.S.A.	0.74
jorropo.net	163.172.31.60	Paris, France	0.76
jorropo.ovh	51.75.127.200	Roubaix, France	0.76
ipfs.stibarc.com	74.140.55.163	Delaware, U.S.A	0.81
ipfs.sloppyta.co	51.68.154.205	Warsaw, Poland	0.83

suggests the value of identifying the fast IPFS gateways in AGChain for efficient app downloading (see §3.4).

7 DISCUSSION

In this section, we discuss several potential improvements AGChain could integrate with in the future and a few other potential extensions.

Deployment challenges. One particular challenge in the real-world deployment of AGChain is inviting developers to participate in the platform. Although AGChain currently supports real-time app delegation for users, it would be more advantageous if a wider range of apps could be stored on the platform in advance to improve user convenience. However, engaging developers to contribute their apps to AGChain requires proactive outreach efforts, such as using publicly available contact information on Google Play to send invitation emails. Moreover, incentivizing developers, such as offering to cover uploading gas fees, is necessary to reduce barriers to adoption, which adds complexity and cost to the deployment process.

More secure. Although AGChain has implemented repackaging checks to detect repackaged apps, it currently lacks the ability to detect general Android malware. To address this limitation, we have devised a plan to integrate the malware scan feature of VirusTotal [18]. By utilizing VirusTotal APIs, we will scan each uploaded app on the server side, and the results of these scans will be stored as part of the app metadata within our smart contract, including a URL link to the scan report. This enhancement will allow users to verify the security of apps during the downloading process by reviewing the scan reports. Additionally, explicit warnings for suspicious apps will be provided to enhance user security and trust. Although these measures offer significant protection, we acknowledge that advanced threats like Advanced Persistent Threats (APTs) require a broader approach involving multiple defensive mechanisms and continuous monitoring, which is beyond the scope of this work. Our contribution focuses on strengthening app integrity through repackaging detection and integration of well-established malware scanning tools, providing a secure environment for typical threat scenarios that users might encounter.

More user-friendly. In our current AGChain prototype, users are required to download apps by specifying package names and version numbers, which may be cumbersome for individuals with limited technical expertise. To enhance user experience, we plan to develop a more intuitive interface that allows users to input or search for app names directly. We could achieve this by crawling app page information from existing app markets and maintaining a server-side table that maps app names (in different languages) to their corresponding package names, ensuring users can easily find desired apps without needing to remember complex identifiers. Since app names are not unique identifiers like package names, there is no need to store them in the smart contract. This enhancement will benefit both app uploaders and downloaders: for app uploaders, a guided uploading process will provide step-by-step assistance to reduce errors and facilitate easier onboarding; for app downloaders, improved search functionality will make app discovery more accessible while also enhancing security by mitigating the risk of adversaries using deceptive, slightly different package names to distribute fake apps. Overall, these improvements will contribute to a more user-friendly and secure ecosystem for all users.

Other potential extensions. In this paper, we implemented AGChain on the current Ethereum ecosystem, but it could be further extended in terms of supporting legacy blockchains [52] and concurrent smart contract execution [53]. Moreover, our crowdsourced server nodes could be enhanced by integrating TEE SGX [54] and becoming more serverless [21]. Additionally, the security and upgradeability of AGChain's smart contract could be protected by some of the state-of-the-art Ethereum protection techniques [39, 40, 58].

8 RELATED WORK

In this section, we present some other works that are closely related to AGChain.

AGChain is mostly related to several recent works [31, 33, 38, 41, 42, 47, 49, 60] that also leveraged the blockchain and IPFS technology to construct decentralized systems in various domains. For example, PubChain [47] is a decentralized publication platform that stored paper metadata in the blockchain layer and raw paper files in IPFS. It introduced an incentive mechanism called PubCoin, which rewarded participants through a process referred to as "publishing or reviewing as mining". Another paper on arXiv, DClaims [41], presented a censorship-resistant service that utilized decentralized web notations to disseminate information on the Internet. Similar to AGChain, it used Ethereum as the blockchain platform and integrated IPFS as the backend data storage. Considering the frequency of web notations among numerous users, DClaims established a small network of nodes to aggregate multiple blockchain transactions and broadcast them collectively, thereby reducing the average transaction cost. Apart from these works, Shafagh et al. [42] published a pioneer study on integrating blockchain and IPFS, drawing inspiration from the four-layer design of the Blockstack [23] system: blockchain, virtualchain, routing, and storage layers. Furthermore, Wei et al. [49] proposed a scalable and trustworthy infrastructure for collaborative container repositories, which is highly pertinent to our work on distributed infrastructures. Klaine et al. [33] presented a privacy-preserving blockchain platform designed for data marketplaces, offering valuable insights into secure data sharing mechanisms. Zieglmeier et al. [60] discussed decentralized inverse transparency using blockchain technology, which complements our focus on enhancing transparency and trust in distributed systems. In comparison to these related studies, AGChain stands out due to its unique characteristics in the following four aspects:

Firstly, unlike other blockchain systems that require users to choose between existing IT infrastructure and their own, our goal with AGChain was not to replace existing app markets. Instead, we designed AGChain as a gateway that not only provides permanent app delegation to end users but also leverages the vast number of apps available in existing markets. We believe that this design approach offers a unique opportunity to combine the benefits of traditional IT infrastructure with decentralized blockchain technology.

Secondly, we dramatically reduced gas costs in AGChain by proposing a set of design-level mechanisms. In contrast, no aforementioned related works [38, 41, 42, 47] tried to do that. While there have been other works focusing on gas optimization, they primarily focused on language-level optimizations. Specifically, GASPER [28] identified gas-costly smart contract coding patterns and summarized them into two categories, loop-related and useless codes. MadMax [30] leveraged control- and data-flow analysis of smart contracts' bytecode to detect the gas-related vulnerabilities, including unbounded mass operations, non-isolated external calls, and integer overflows. Additionally, GASOL [22] introduced a gas optimization approach by replacing multiple accesses to the global storage data with several single accesses to the data in local memory. Accessing local memory incurs significantly fewer gas costs (3 Gas per access) compared to accessing storage data (each write access costing 20 Gas in the worst case and 5 Gas in the best case). However, these optimizations are still limited to the code level and are specifically designed for particular gas-costly patterns.

Thirdly, to the best of our knowledge, none of the existing studies on IPFS [27, 38, 41, 42, 47] explicitly mentioned the undistributed problem of IPFS, where IPFS files are cached in other nodes only when the node requests that file. In addition to our attempt to this problem in §3.4, the IPFS designers themselves also tried to address this issue. Recently, they have launched Filecoin [8], which is an incentive token mechanism aimed at encouraging peers in the IPFS network to remain online and take responsibility for storing files. However, Filecoin requires users to pay for file storage, and the process of storing a 1MB file currently takes five to ten minutes in the Filecoin network [8]. Due to these limitations, we built our own IPFS consortium network that leveraged IPFS gateways and crowdsourced server nodes to periodically backup files.

Lastly, we encountered three context-specific challenges that are unique to AGChain, as explained in §4.

9 CONCLUSION AND FUTURE WORK

In this paper, we proposed AGChain, a blockchain-based gateway for trustworthy – permanent, distributed, and secure – app delegation from existing app markets. We addressed challenges in significantly reducing smart contract gas costs and enabling fully distributed IPFS-based file storage. Additionally, we resolved three specific system issues for security and sustainability. We implemented an AGChain prototype on Ethereum and Polygon blockchains, evaluating its performance, gas costs, security, and decentralization. The results showed a 12% performance overhead and a cost of around 0.0028 USD per app upload (no cost for app download). Further improvements include making AGChain more developer-friendly and secure.

Future work could explore integrating AGChain with other blockchain platforms to enhance interoperability and expand its reach beyond existing app markets. This integration could allow for broader use cases, such as cross-chain app delegation and support for diverse application ecosystems. Additionally, extending the functionalities of AGChain beyond app markets, such as facilitating secure software updates or supporting decentralized content distribution, presents promising avenues for further research and development in the blockchain space. We will also consider incorporating user feedback analysis and additional benchmarks in the future.

REFERENCES

[1] 2016. logs - How does Ethereum make use of bloom filters? https://ethereum.stackexchange.com/questions/3418/how-does-ethereum-make-use-of-bloom-filters/.

[2] 2017. Change Your App Store Country to Download Region-Locked Apps & Games on Your iPhone. https://tinyurl.com/cehv4cxy.

[3] 2018. Diving into Ethereum's world state. https://medium.com/cybermiles/diving-into-ethereums-world-state-c893102030ed.

[4] 2018. Easily Change Your Play Store Country to Download Region-Locked Apps & Games. https://tinyurl.com/wbbt96ea.

[5] 2018. ETH Bloom Filters. https://medium.com/@naterush1997/eth-goes-bloom-filling-up-ethereums-bloom-filters-68d4ce237009.

[6] 2018. Top 5 most dangerous Public WIFI attacks. https://e-channelnews.com/top-5-most-dangerous-public-wifi-attacks/.

[7] 2019. 5 Best VPNs to watch TVB from anywhere. https://www.comparitech.com/blog/vpn-privacy/bvpn-tvb/.

[8] 2020. Filecoin Network Performance. https://docs.filecoin.io/about-filecoin/network-performance/.

[9] 2020. How to access Google Play Store in China. https://www.bestvpn.co/how-to-unblock-websites/access-google-play-store-china/.

[10] 2020. How to Unblock Hulu from Anywhere. https://www.vpnmentor.com/blog/how-to-unblock-hulu-from-anywhere/.

[11] 2020. JS IPFS. https://js.ipfs.io/.

[12] 2020. web3.js for Ethereum. https://web3js.readthedocs.io/en/v1.3.0/.

[13] 2020. web3.py for Python. https://web3py.readthedocs.io/en/stable/.

[14] 2021. Androguard. https://androguard.readthedocs.io/.

[15] 2021. Mitmproxy. https://mitmproxy.org/.

[16] 2021. Polygon: Ethereum's Internet of Blockchains. https://polygon.technology/lightpaper-polygon.pdf.

[17] 2021. The Power of Content-addressing. https://flyingzumwalt.gitbooks.io/decentralized-web-primer/content/avenues-for-access/lessons/power-of-content-addressing.html.

[18] 2021. VirusTotal. https://www.virustotal.com/.

[19] 2023. App Backup - Easy and Fast! Su. https://play.google.com/store/apps/details?id=com.colure.app.ibu.

[20] 2023. Apps Backup and Restore. https://play.google.com/store/apps/details?id=com.touchfield.appbackuprestore.

[21] Angeliki Aktypi, Dimitris Karnikis, Nikos Vasilakis, and Kasper Rasmussen. 2022. Themis: A Secure Decentralized Framework for Microservice Interaction in Serverless Computing. In *Proc. ACM Conference on Availability, Reliability and Security (ARES)*.

[22] Elvira Albert, Jesús Correas, Pablo Gordillo, Guillermo Román-Díez, and Albert Rubio. 2020. GASOL: Gas Analysis and Optimization for Ethereum Smart Contracts. In *Proc. Springer TACAS*.

[23] Muneeb Ali, Jude Nelson, Ryan Shea, and Michael J. Freedman. 2016. Blockstack: A Global Naming and Storage System Secured by Blockchains. In *Proc. USENIX ATC*.

[24] Kevin Allix, Tegawende F. Bissyande, Jacques Klein, and Yves Le Traon. 2016. AndroZoo: Collecting Millions of Android Apps for the Research Community. In *Proc. ACM MSR*.

[25] Juan Benet. 2014. IPFS - Content Addressed, Versioned, P2P File System. *CoRR arXiv* abs/1407.3561 (2014).

[26] Vitalik Buterin et al. 2014. A next-generation smart contract and decentralized application platform. *white paper* (2014).

[27] Fran Casino, Eugenia Politou, Eethimios Alepis, and Constantinos Patsakis. 2020. Immutability and Decentralized Storage: An Analysis of Emerging Threats. *IEEE Access* 8 (2020).

[28] Ting Chen, Xiaoqi Li, Xiapu Luo, and Xiaosong Zhang. 2017. Under-optimized smart contracts devour your money. In *Proc. IEEE SANER*.

[29] Brian Curran. 2018. What is Interplanetary File System IPFS? Complete Beginner's Guide.

[30] Neville Grech, Michael Kong, Anton Jurisevic, Lexi Brent, Bernhard Scholz, and Yannis Smaragdakis. 2018. MadMax: surviving out-of-gas conditions in Ethereum smart contracts. In *Proc. ACM OOPSLA*.

[31] Songlin He, Eric Ficke, Mir Mehedi Ahsan Pritom, Huashan Chen, Qiang Tang, Qian Chen, Marcus Pendleton, Laurent Njilla, and Shouhuai Xu. 2022. Blockchain-based automated and robust cyber security management. *J. Parallel Distributed Comput.* 163 (2022), 62–82.

[32] Rahul Hiran, Niklas Carlsson, and Phillipa Gill. 2013. Characterizing Large-scale Routing Anomalies: A Case Study of the China Telecom Incident. In *Proc. Springer PAM*.

[33] Paulo Valente Klaine, Hao Xu, Lei Zhang, Muhammad Imran, and Ziming Zhu. 2023. A Privacy-Preserving Blockchain Platform for a Data Marketplace. *ACM Distributed Ledger Technologies: Research and Practice* 2, 1 (2023).

[34] Li Li, Tegawendé F. Bissyandé, and Jacques Klein. 2019. Rebooting Research on Detecting Repackaged Android Apps: Literature Review and Benchmark. *IEEE TSE* 47, 4 (2019).

[35] James Lloyd. 2020. What is next for Asia in FinTech adoption. https://www.ey.com/en_gl/banking-capital-markets/what-is-next-for-asia-in-fintech-adoption.

[36] Haoyu Ma, Shijia Li, Debin Gao, Daoyuan Wu, Qiaowen Jia, and Chunfu Jia. 2021. Active Warden Attack: On the (In)Effectiveness of Android App Repackage-Proofing. In *IEEE Transactions on Dependable and Secure Computing*.

[37] Jude Nelson, Muneeb Ali, Ryan Shea, and Michael J. Freedman. 2016. Extending Existing Blockchains with Virtualchain. In *Workshop of Distributed Cryptocurrencies and Consensus Ledgers*.

[38] Van-Duy Pham, Canh-Tuan Tran, Thang Nguyen, Tien-Thao Nguyen, Ba-Lam Do, Thanh-Chung Dao, and Binh Minh Nguyen. 2020. B-Box - A Decentralized Storage System Using IPFS, Attributed-based Encryption, and Blockchain. In *Proc. IEEE RIVF*.

[39] Michael Rodler, Wenting Li, Ghassan O. Karame, and Lucas Davi. 2021. EVMPatch: Timely and Automated Patching of Ethereum Smart Contracts. In *Proc. USENIX Security Symposium*.

[40] Mehdi Salehi, Jeremy Clark, and Mohammad Mannan. 2022. Not so immutable: Upgradeability of Smart Contracts on Ethereum. *CoRR* abs/2206.00716 (2022).

[41] João Santos, Nuno Santos, and David Dias. 2019. DClaims: A Censorship Resistant Web Annotations System using IPFS and Ethereum. *CoRR arXiv* abs/1912.03388 (2019).

[42] Hossein Shafagh, Lukas Burkhalter, Anwar Hithnawi, and Simon Duquennoy. 2017. Towards Blockchain-based Auditable Storage and Sharing of IoT Data. In *Proc. ACM Cloud Computing Security Workshop*.

[43] Petar Tsankov, Andrei Marian Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Bünzli, and Martin T. Vechev. 2018. Securify: Practical Security Analysis of Smart Contracts. In *Proc. ACM CCS*.

[44] Sarah Underwood. 2016. Blockchain Beyond Bitcoin. *Commun. ACM* 59 (2016).

[45] Haoyu Wang, Zhe Liu, Jingyue Liang, Narseo Vallina-Rodriguez, Yao Guo, Li Li, Juan Tapiador, Jingcun Cao, and Guoai Xu. 2018. Beyond Google Play: A Large-Scale Comparative Study of Chinese Android App Markets. In *Proc. ACM IMC*.

[46] Huibo Wang, Pei Wang, Yu Ding, Mingshen Sun, Yiming Jing, Ran Duan, Long Li, Yulong Zhang, Tao Wei, and Zhiqiang Lin. 2019. Towards Memory Safe Enclave Programming with Rust-SGX. In *Proc. ACM CCS*.

[47] Taotao Wang, Soung Chang Liew, and Shengli Zhang. 2020. PubChain: A Decentralized Open-Access Publication Platform with Participants Incentivized by Blockchain Technology. In *Proc. IEEE ISNCC*.

[48] Yibo Wang, Qi Zhang, Kai Li, Yuzhe Tang, Jiaqi Chen, Xiapu Luo, and Ting Chen. 2021. iBatch: Saving Ethereum Fees via Secure and Cost-Effective Batching of Smart-Contract Invocations. In *Proc. ACM ESEC/FSE*.

[49] Franklin Wei, Stephen Tate, Mahalingam Ramkumar, and Somya Mohanty. 2022. A Scalable Trustworthy Infrastructure for Collaborative Container Repositories. *ACM Distributed Ledger Technologies: Research and Practice* 1, 1 (2022).

[50] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* (2014), 1–32.

[51] Daoyuan Wu, Debin Gao, and David Lo. 2021. Scalable online vetting of Android apps for measuring declared SDK versions and their consistency with API calls. *Empirical Software Engineering* 26, 1 (2021).

[52] Karl Wüst, Loris Diana, Kari Kostiainen, Ghassan Karame, Sinisa Matetic, and Srdjan Capkun. 2021. Bitcontracts: Supporting Smart Contracts in Legacy Blockchains. In *Proc. ISOC NDSS*.

[53] Karl Wüst, Sinisa Matetic, Silvan Egli, Kari Kostiainen, and Srdjan Capkun. 2020. ACE: Asynchronous and Concurrent Execution of Complex Smart Contracts. In *Proc. ACM CCS*.

[54] Yang Xiao, Ning Zhang, Jin Li, Wenjing Lou, and Y. Thomas Hou. 2020. PrivacyGuard: Enforcing Private Data Usage Control with Blockchain and Attested Off-Chain Contract Execution. In *Proc. Springer ESORICS*.

[55] Xiao Yi, Yuzhou Fang, Daoyuan Wu, and Lingxiao Jiang. 2023. BlockScope: Detecting and Investigating Propagated Vulnerabilities in Forked Blockchain Projects. In *Proc. ISOC NDSS*.

[56] Xiao Yi, Daoyuan Wu, Lingxiao Jiang, Yuzhou Fang, Kehuan Zhang, and Wei Zhang. 2022. An empirical study of blockchain system vulnerabilities: modules, types, and patterns. In *Proc. ACM FSE*.

[57] Haoqian Zhang, Yancheng Zhao, Abhishek Paryani, and Ke Yi. 2020. Infnote: A Decentralized Information Sharing Platform Based on Blockchain. *CoRR arXiv* abs/2002.04533 (2020).

[58] Yuyao Zhang, Siqi Ma, Juanru Li, Kailai Li, Surya Nepal, and Dawu Gu. 2020. SMARTSHIELD: Automatic Smart Contract Protection Made Easy. In *Proc. IEEE SANER*.

[59] Wu Zhou, Yajin Zhou, Xuxian Jiang, and Peng Ning. 2012. Detecting Repackaged Smartphone Applications in Third-Party Android Marketplaces. In *Proc. ACM CODASPY*.

[60] Valentin Zieglmeier, Gabriel Loyola Daiqui, and Alexander Pretschner. 2023. Decentralized Inverse Transparency with Blockchain. *ACM Distributed Ledger Technologies: Research and Practice* 2, 3 (2023).

Received 31 December 2023; revised 17 October 2024; accepted 1 December 2024