Journal of Computing and Security

Research Article, 2022, Volume 9, Number 2 (pp. 39–54)

http://www.jcomsec.org

Decentralized App Store and License Management Using Smart Contracts and Self-Sovereign Identities

Sadegh Dorri Nogoorani [a,*]

[a] *Faculty of Electrical and Computer Engineering, Tarbiat Modares University, Tehran P.O. Box 14115-111, Iran.*

ARTICLE INFO.

Article history:
Received: 16 July 2022
Revised: 30 August 2022
Accepted: 15 October 2022
Published Online: 18 December 2022

Keywords:
App Store, Blockchain, Smart Contract, Software License Management, Decentralized Identifier, Self-Sovereign Identity, Non-Fungible Token (NFT).

ABSTRACT

Mobile applications are playing an important role in our digital lives. App stores can be considered key components in the ecosystem of mobile applications. They assist users to ensure the authenticity of applications and protecting the intellectual property rights of application developers. In this paper, we introduce an autonomous decentralized mobile application distribution platform (app store) and a license management solution that utilizes a public blockchain and operates by smart contracts. We identify developers by their decentralized and self-sovereign identities, verify the integrity of the applications according to the secure information on the blockchain, and implement a fully autonomous license management solution by non-fungible tokens (NFT) on the blockchain. We deploy a proof-of-concept implementation of our proposal written in Solidity language on the Ropsten (Ethereum) and RSK testnets, and evaluate its latency and costs. Our comparison with the related works demonstrates that our proposal ranks atop the related works.

1 Introduction

Nowadays, smartphones have become an important part of our daily lives, and with the use of applications installed on them, day-to-day work is done seamlessly and effortlessly. Many businesses offer services solely through mobile applications, and a variety of digital multimedia content, ranging from text, images, movies, and series to mobile games, are provided on smartphones. This diversity of content as well as the need to publish proprietary versions of mobile features have led to the release of software through intermediary distribution and publishing platforms called *app stores*. An app store features a repository for users to download, install and update different applications, and eliminates some of the problems in the release of applications. Moreover, app stores typically aid application developers in addressing some other common problems which need to be addressed by a supporting platform, and include services for selling licenses, in-app payments, in-app advertisements, ratings and user surveys, etc[1].

Traditional app stores, such as Google Play Store [1] and Cafe Bazaar [2], are centrally operated by the companies which develop them, and are under the control of those companies. Centralized app stores usually have more-rigid identification procedures for developers, and stricter compliance rules that potentially bring about some benefits for user experience, protection against fraud, and detection of fake/harmful

* Corresponding author.
Email address: dorri@modares.ac.ir (S. Dorri Nogoorani)
https://dx.doi.org/10.22108/JCS.2022.134436.1103
ISSN: 2322-4460

[1] https://play.google.com/store/
[2] http://cafebazaar.ir/

applications.

However, app stores have not been completely successful in this regard, and, once in a while, malicious applications cross these barriers.

Centralization and intermediation have put obstacles in the way of software developers on the one hand and users on the other. App store restriction policies, demand for sales commission, delay in settlement, and slowness in releasing updates provided by developers are some of these cases. As an example, in 2017, Apple's App Store excluded all Iranian applications with financial facilities under the pretext of US sanctions [2]. These shortcomings are not specific to mobile app stores and apply to other centralized digital content distribution platforms such as Amazon or YouTube, as well [3]. Some app stores, such as F-Droid [3] and Aptoide [4] have a decentralized publishing model. In these app stores, some applications may be featured in the main repository and be controlled by project maintainers. The applications distributed through third-party repositories are still controlled by their respective companies. Aptoide also supports an ERC-20 token for in-app purchases [5].

One of the challenging problems in a decentralized app store is how to prevent piracy and protect intellectual property rights. Crussell et al. developed tools and analyzed android applications in this respect. According to their study in 2012 which included 75,000 Google Play Store applications, 141 cloned applications were identified [4]. In another study in 2013, 4,295 cloned applications were identified among 265,239 downloaded applications [5]. These results demonstrate that the protection of intellectual properties is a real issue in app stores. By using these tools one can recognize the similarity of applications, but to mark an application as real or cloned, one has to refer to the history of the application's publication. However, this information is not conclusively available in existing app stores, and older versions of an application may not be kept by the app store, for a long time.

With the increasing popularity of Bitcoin, blockchain technology has flourished and gained a lot of attention both in industry and academia, and researchers are using it to address the complex issues of traditional processes. Blockchain is maintained and updated by a distributed network of autonomous nodes around the world. For this reason, a single point of failure is prevented, and through consensus between nodes, they all have a single view of the information stored on the blockchain and can protect it from manipulation. Therefore, while delivering decentralization and distribution, blockchain is a suitable infrastructure for document registration, fraud prevention, and secure code deployment [6]. In particular, if we store software code on a blockchain, it is available to all interested parties (*single source of truth*) and is protected from manipulations. There is also promising progress toward using blockchain technology for protecting intellectual property rights [7].

In this paper, we propose a decentralized app store implemented by autonomous smart contracts. The application files are stored on a cloud file storage of the developer's choice. Application licenses are stored and managed on the blockchain. Therefore, even if the software company becomes bankrupt, or discontinues supporting the software, the license exists on the blockchain and the user can use it.

On one side, the developers can use our proposed app store to publish applications and sell licenses without any restrictions. On the other side, application users can rely on our proposed app store to receive authentic software, activate their license, and even transfer a license to other users in a peer-to-peer manner. Both sides are directly connected without any intermediary. Our contributions in this paper are as follows:

- The developers are authenticated by public key cryptography and Decentralized Identifiers (DIDs). Henceforth, we natively support Self-Sovereign Identities (SSI).
- Each application license is a unique token bound to the installation device through public key cryptography.
- License purchase is offered in an atomic and autonomous manner. In other words, the price of the license is transferred from the user to the developer's address, and in the same blockchain transaction, a fresh new license is minted and transferred to the user.

The rest of the paper is organized as follows: Section 2 presents a background on the concepts used in our paper; Section 3 reviews the related works; Section 4 describes our proposed solution; and Section 5 is devoted to the evaluation of our proposal and comparison with the related work. Finally, Section 6 concludes the paper.

2 Background

We briefly explain the foundations of our proposal in this section. In particular, we give a background on software protection mechanisms and the typical software purchase flow. Then, we introduce the concepts

[3] https://f-droid.org/
[4] http://aptoide.com/
[5] https://appcoins.io/

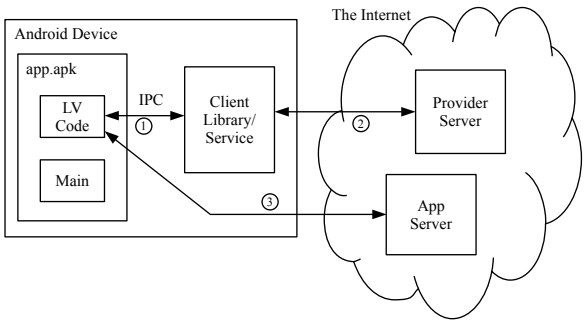

Figure 1. Traditional License Validation Process.

of blockchain, smart contracts, and decentralized identifiers.

2.1 Software Protection

Historically, software vendors focused on protecting their software code from being illegally downloaded or copied. Once a user was able to copy the software and the protection mechanism was circumvented, nothing could be done to stop a pirate copy. Nowadays nearly all users are connected to the Internet, the protection mechanisms have evolved into online activation and licensing methods.

Nonetheless, the code shall be protected from reverse engineering and hacking which disable the license validation routines. The fact that the software is run on a platform that is under the control of its user, makes it impossible for the software alone to verify its integrity and authenticity and detect these hacks.

Mobile device and operating system vendors enforce compliance rules, and restrict third-party installation sources, hoping to prevent pirate and malicious applications from installation. However, these precautions do not stop enterprises from disobeying licensing conditions. In addition, they cannot prevent non-compliant devices and users who bypass standard operating system limitations (e.g., by *rooting* or *jail breaking*) from installing pirate applications. The interested reader is referred to [8] for a fundamental discussion, and [9] for a good review of software protection methods.

2.2 Typical Software Purchase Flow

Support for purchases is an important feature of an app store. Users of the app store may purchase an application license in the first place, or buy in-app products/subscriptions later on. While the exact details are different for each app store, they follow almost the same concepts, which we will explain in this section.

Nonetheless, the developer follows the following steps:

1) Configures the app store (provider).
2) Adds some features to the app which require licensing.
3) Inserts a verification code. The code may depend on app store-provided libraries and/or client-side services.
4) Uses a code/binary obfuscator to prevent reverse engineering techniques and code modifications that disable the verification code.
5) Publishes the application.

The user follows the following steps to purchase and run a paid application:

1) Selects the application for purchase.
2) Completes the payment in the app store.
3) Installs the application (and the required services, if applicable.)
4) Runs the application.
5) The application verifies the license before offering critical functionalities.

Note that some applications offer limited functionality without buying the license. In this case, the application is also installable before payment. The validation process is an essential part of all the mentioned steps. Figure 1 illustrates the traditional way of the validation process:

1) The application code calls a local client library/service (IPC).
2) The client connects to the app store server, and hands in the required application, user, and device information to the server. The server verifies the purchase and responds to the client.
3) Optionally, the application connects to its server and verifies the received response from the client on the server side.

The last step (3) is recommended and makes it harder for a malicious user to circumvent the validation process.

There is another way in which the application only connects to its server, and the server connects to the app store server. In this case, some information about the device and its user may not be accessible to the

application to be provided to the server.

2.3 Blockchain

The term *blockchain* is used to refer to either of the following parts of a *blockchain system* [10]:

- A distributed *network* of *nodes* connected to each other in an *overlay network*.
- A distributed *data structure* which is collectively maintained by the blockchain nodes. This structure is a sorted list of consecutive blocks each of which contains some transactions. The more-general term *distributed ledger* is also used to accompany other non-linear collections (e.g., a directed acyclic graph (DAG)) of transactions.
- A distributed *network protocol* for maintaining the overlay network and the data structure as well as defining incentive mechanisms (particularly in public blockchains). All the rights and responsibilities of the nodes and their communications are defined by the network protocol. In addition, verification, validation, and consensus across the nodes in the network are specified by the protocol.

Every time a new transaction occurs on the blockchain, a record of that transaction is added to every participant's ledger. In a public blockchain, all participating nodes verify and agree on the validity of transactions and the order of the blocks. These agreements are reached by consensus algorithms. These algorithms ensure that protocol rules are followed. Hence, all operations are performed in a manner that does not require trust. Several types of consensus algorithms are used in various blockchains, e.g. proof of work, proof of stake, and delegated proof of stake. Some blockchains use a combination of two or more consensus methods to take advantage of each method and avoid its disadvantages [11].

Blockchain users manage their assets with their digital wallet of choice.

The digital wallet keeps track of the user's cryptocurrency (such as Bitcoin and Ether) and performs other interactions with the blockchain. A simple cryptocurrency wallet generates pairs of public and private cryptographic keys and stores them in a secure storage [11]. Therefore, the user's private key is not stored in a central server or anywhere on the Internet, and is kept locally on the user's device. The public key is used to create an address for the user which is similar to a bank account number and can be used to receive cryptocurrency.

2.4 Smart Contract

Some public blockchains support smart contracts. Ethereum is the second most well-known public blockchain and the most popular smart contract platform. A smart contract is a program code (often small) that can control an amount of cryptocurrency, and take actions according to the terms specified in its code. Smart contracts are stored on the blockchain, and their correct execution is guaranteed by the blockchain platform. Smart contracts provide a platform for Decentralized Autonomous Organizations (DAOs). These organizations operate automatically by the rules defined in computer programs and smart contracts. A DAO can be trusted because it is transparent and always follows a pre-agreed algorithm that cannot be manipulated. In contrast, one should trust a traditional organization because it is government-led or the like.

Solidity [6] is one of the most popular programming languages for smart contracts. It is an object-oriented, high-level language that is influenced by Python, C++ and JavaScript languages and is designed for the Ethereum Virtual Machine (EVM). Solidity supports inheritance, libraries, and complex user-defined types. There are different methods for applications to connect to a blockchain. For example, MetaMask cryptowallet [7] is a browser extension that allows web pages to use Javascript for connecting to the wallet and interacting with any blockchain that exposes an Ethereum-compatible JSON RPC API. The user's private key is not passed to the web application, and the wallet signs the transactions after receiving a confirmation from the user.

Processing transactions on a public blockchain imposes costs on blockchain nodes. To prevent free-riding and to better protect the blockchain from useless transactions, these blockchains require the sender of a transaction to pay some processing/transaction fee. The amount of fee may be fixed, or depend on the underlying operations and their relative cost. In Ethereum, transaction fees are measured in terms of *gas*. Each opcode in the EVM has a specific gas cost, and the cost of each transaction is determined according to the amount of gas needed to execute its opcodes. The price of gas is a dynamic value and is dependent on the supply and demand mechanism. Once a user issues a transaction, the required amount of gas is deduced from his balance, according to the gas price (in Ether).

Public blockchains enable everyone to use the blockchain to publish a new token, manage its supply, record its owner, and transfer tokens to other users.

[6] `https://soliditylang.org/`
[7] `https://www.metamask.io/`

Most tokens are fungible, meaning that they all are the same and one can be replaced by another. Some other tokens are non-fungible, in the sense that they are unchangeable, different, and unique. In this paper, we use a non-fungible token (NFT) for the application license. More details are given subsequently.

2.5 RSK Blockchain

RSK [8] (formerly known as RootStock Platform) is an Ethereum-compatible smart contract platform that is powered by Bitcoin. In particular, it is a Bitcoin side-chain with two-way pegging to Bitcoin and SHA-256D merged mining. RSK has its ledger and network of nodes, however, its miners mine transactions for both networks. The native cryptocurrency of the RSK network (RBTC) is not issued on its own, and all RBTC tokens are coming from the Bitcoin network. Bitcoins can be transferred to the RSK network to be accessible as RBTC on the network. Afterward, they can be used to pay RSK network fees or transferred to other addresses. It is possible at any time to transfer RBTC tokens back to the Bitcoin network at no additional cost other than the standard RSK and Bitcoin transaction fees. RSK has faster confirmation times compared to Bitcoin, and lower transaction costs compared to Ethereum.

2.6 Decentralized Identifiers

We rely on *Decentralized Identifiers* [12] (DIDs) to identify developers in our proposed app store. DIDs are a means to support *Self-Sovereign Identities* (SSI) [13]. SSI is the most recent approach in identity management in which the owner has full control over his/her identity. Each DID consists of a method name and some method-specific data. For example, `did:ethr:0xf3be...0d74` corresponds to an EtherDID [14] originally owned by the Ethereum mainnet account `0xf3be...0d74`. Therefore, no registration is required to claim an identity in the first place. Each DID can have one or several attributes to be better introduced, and these attributes can be signed by respective authorities to be verifiable (Verifiable Claims/Credentials).

EtherDIDs are also implemented on similar public blockchains other than Ethereum (mainnet and testnets), namely RSK, Alastria Telsius, and ARTIS networks. While decentralized identifiers are not confined to EtherDIDs, without loss of generality, we aimed to rely on EtherDIDs because they can be used by smart contracts on these Ethereum-like blockchains for identification and authorization purposes. Any other on-chain DID method can be used on other blockchains to implement our proposed app store. In the following, we present a brief explanation of how EtherDIDs work. By default, an identity is *controlled* by whoever has the corresponding private key (original owner). However, it is also possible for the controller to transfer this role to another external or smart contract account.

This functionality opens up possibilities for key rotation and multiple ownership (multi-signature) scenarios. In addition, the identity controller can designate one or multiple *delegates* to perform tasks on his/her behalf. All the mentioned details about identity are stored on and publicly accessible from the blockchain.

2.7 InterPlanetary File System (IPFS)

The InterPlanetary File System (IPFS) [15] is a well-known peer-to-peer cloud file storage which is used in many blockchain-based systems. The content-addressable nature of IPFS makes it easy to use and compatible with the way transactions and blocks are identified in blockchains such as Bitcoin and Ethereum. In particular, all files and storage blocks in IPFS are immutable, and their cryptographic hash (e.g., SHA-2) can be used to retrieve them. IPFS content is replicated among several nodes and can be accessed even if the owner of the data is temporarily down.

3 Related Works

Blockchain technology has already been used in different areas related to app stores. For example, it can be used to collect users' feedback on applications provided in an app store [16]. The blockchain offers a decentralized and non-proprietary trusted storage to record user rankings or mark an app for further security-related investigations. Different app stores can send the users' feedback on applications to one blockchain smart contract, and base their reputation calculations on the unified recorded information by the contract.

The general idea that a blockchain can be used for software licensing does not only belong to this paper. For example, the Deco.Network collection of smart contracts has a license selling feature [17] which is a simple registry for license sales. Another proof-of-concept is presented in [18]. In this case, the licenses are implemented by ERC-20 tokens. The user has to sign an on-chain challenge to prove ownership of the token. In [19] an ERC-721 token is used to implement software licenses. Upon purchasing a license from the website of an application, a fresh token is transferred to the user. The token is *inactive* until the user enters the tokenID in the application. The application contacts

[8] `https://www.rsk.co/`

the token contract and flags the token as *activated* (requires a transaction). A token can only be activated once, and during the activation, it is bound to the device identifier to be useful only for that device. However, there are also well-founded proposals in the academic literature which we subsequently review.

Stepanova and Eriņš [20] have conducted a study and assessed the opportunities and prospects for the use of blockchain in software licensing. They explained the functionalities of individual components of a software license transfer system. In their generic model, a smart contract is provisioned to enforce licensing conditions and keep a link to a repository for downloading the application files.

Once the user accepts the conditions, pays the license price in cryptocurrency, and is subsequently permitted to use the application. The details of the purchase are recorded on the blockchain for further use.

Master Bitcoin Model is a basic form of software license validation proposed by Fortin [21] and implemented by Lebo in the Dissent project using the Namecoin blockchain [9] . In this model, the existence of a Bitcoin sent from the vendor's address in the user's address indicates the ownership of the software. In other words, if a user has a transaction that transfers a Bitcoin from a particular vendor address, that user is considered to be legitimate.

Herbert and Litchfield proposed the Bespoke Model [22] and based the ReSOLV system [9] on it. ReSOLV is a decentralized software license validation system that uses a customized version of the Bitcoin blockchain. A special transaction is defined with license-related fields to entitle a user to use a particular software (see Figure 2.) The user address that has the token issued by the vendor is entitled to use the software. The software vendor uses the user's public key to encrypt some of the data in these fields and signs the transaction by its private key. For the software to be executed, some bootstrap code which is stored on the blockchain is required to prevent unauthorized versions. The bootstrap is part of the executable code of the software that is used to pre-execute the software or implement a critical part of the program. The bootstrap may be modified with every software update, to make it difficult to be reverse-engineered. The bootstrap code is not persisted on the user's system storage and should be downloaded and decrypted in every run of the application. Therefore, the Bespoke Model can be used to implement flexible licensing models such as short-term subscriptions.

The idea of an open-source decentralized application marketplace was also proposed in Spheris white paper [23]. Spheris aimed to create a place for developers to take control of their revenue, and for consumers to pay directly to the developers. The platform was based on the Ethereum blockchain, and its *catalog* was to be managed by a smart contract. It maintained metadata about registered developers, applications, subscriptions, and ratings. The file contents were planned to be distributed by integrated third-party content distribution platforms. They provisioned to have a digital asset management interface to the catalog. A feature-extended crypto wallet was proposed to be used for basic Ethereum operations and to manage subscriptions and payments. The general idea of a subscription validation library for Spheris was proposed but its details were not discussed in the white paper. However, the idea was not successful in its crowdfunding and was discontinued on October 2017 [10] .

The realm of software distribution in the Industrial Internet of Things (IIoT) is another context in which blockchain-based solutions are proposed. The blockchain in this case brings about transparency, better tracking, and safe update pushing for software installed on IoT devices. Software maintenance on these devices is different from mobile devices. They do not necessarily have a user-friendly interface and an operator has to attend to each device in its operating environment to manually upgrade its software (firmware). Seitz et al. [24] have proposed IIoT Bazaar which is a blockchain-based application marketplace for IIoT devices. Each device is identified in this system by an attached QR code, and the information related to its installed software is fetched from the blockchain to the attending operator's smart device. Only metadata about each application is on-chain, and screenshots, icons, and binary files are stored in a separate space. Three smart contracts were proposed for managing application information, installation history, and payment (an ERC-20 token). A prototype of IIoT Bazaar was implemented on an enterprise blockchain.

Magnanini et al. [25], proposed a unified licensing system for VNF-support devices. Virtual Network Functions (VNF) such as firewall, monitoring, antivirus, and identity management, are implemented by special software (firmware). In this case, Network Software Vendors (NSVs) have various licensing models, and Communication Service Providers (CSPs) use their software. The unified licensing system uses blockchain technology and smart contracts to connect the two untrusted parties (NSVs to CSPs). In particular, the CSPs are not worried about violating license conditions in dynamic workloads, and NSVs are con-

[9] https://github.com/aaron-lebo/dissent

[10] https://spheris.io

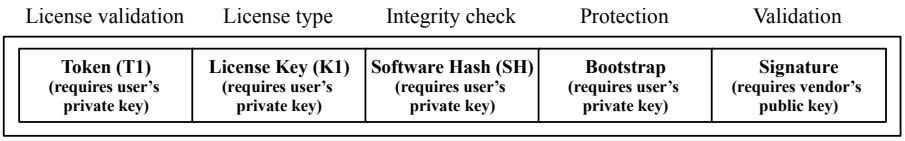

Figure 2. Custom blockchain token in Bespoke Model and ReSOLV system [9, 22].

fident about their income. The blockchain guarantees a secure, inexpensive, and highly available system for this purpose.

Park et al. [26] proposed a decentralized blockchain-based Android app store with the IPFS [15] P2P file system. Upon purchase of an application, the user is allowed to rank the application and its developer by signing the developer's public key. Developers are required to register their application information in a top-level smart contract. Subsequently, the top-level contract deploys an application-specific smart contract and stores the public application signing key of the developer in this contract. Users use this smart contract to purchase the application. The download address is stored in the contract in encrypted form and is sent to the user in an off-chain encrypted message over the Whisper [11] social media application.

The first user of an application is required to download the application files directly from the developer's host. Subsequent users can download the files from other users in a P2P fashion by the IPFS protocol. No mechanism is proposed to stop users from sharing copies of the application between users.

The proposal in this paper is an enhanced version of our autonomous app store which was first introduced in [27]. The former version was implemented on the Ethereum blockchain and operated solely through smart contracts. We also used IPFS in our implementation and stored applications' licenses as non-fungible tokens (NFT) on the blockchain. However, the developers were not authenticated, and there was no way to ascertain either their identity or the true relationship with the code they published in our proposed app store. In addition, licenses were treated lightly in the previous version. In particular, the only protection against piracy was to store some bootstrap code out of the application code. The bootstrap code was encrypted by the public key of the user who purchased the license, and the application needed to download the code and decrypt it with the user's private key in each run. In addition, licenses were not transferrable. A complete comparison with the related works is presented later in the Evaluation Section.

[11] http://whisper.sh/

4 The Proposed Solution

The architecture of our proposed app store is depicted in Figure 3. A smart contract is at the heart of this app store, through which the most critical operations are performed autonomously. These operations, which traditionally require reliance on a trusted third party (the app store company), are carried out on a public blockchain in a decentralized manner. In addition, we have a user interface and store application packages on cloud file storage. We describe these components in the following subsections.

4.1 User Interface (UI)

The user interface is an application through which ordinary users and developers interact with other components of the app store. Although it is possible to interact with and call smart contracts directly, this requires programming knowledge. In addition, an app store has to interact with systems outside the blockchain. The UI is, in fact, a single counter through which users download and purchase applications that developers have uploaded. Also, developers can upload their applications and release updates for them. The user interface does not directly send transactions to the blockchain. Instead, uses an API to connect to the Ethereum wallet under the control of its user. All interactions with the blockchain which require submitting a transaction, are performed with the direct permission of the wallet owner and signed by his/her private key.

More specifically, charging an account with native blockchain cryptocurrency, transferring tokens, registering and publishing applications, purchasing a license, and transferring the license all require approval from the UI's user, and the corresponding transactions are signed and submitted by his/her wallet. By the way, our proposed app store is autonomous and implemented by a smart contract. Therefore, users are not required to use this interface; and they can perform all their tasks directly by interacting with our smart contract and other system components, or even use other (compliant) interfaces for their purposes.

4.2 Cloud File Storage (FS)

Since it is costly to maintain data on the blockchain, the applications are stored outside the blockchain and only their secure hash value is stored on the

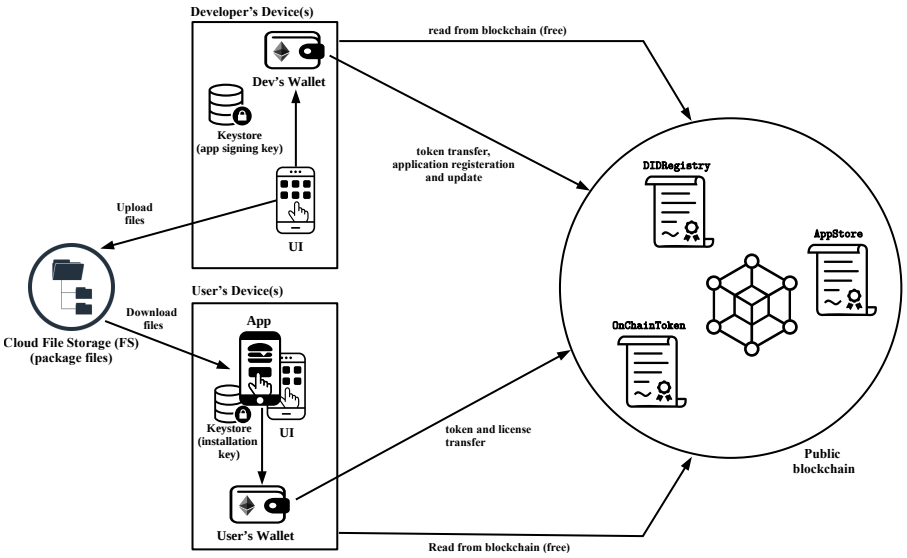

Figure 3. The Proposed App Store Architecture.

blockchain along with other access information in the smart contract. Considering requirements such as high availability, reliability and not having a single point of failure, and good throughput, we need to use a distributed file system [28]. Centralized solutions such as Google Drive or Dropbox, or distributed peer-to-peer (P2P) file-sharing systems such as BitTorrent or IPFS [15] can be used for this purpose.

4.3 The AppStore Contract

This contract is responsible for registering applications and their payment-related information, finding applications, and retrieving compatibility information about them. Meanwhile, the actual application files are stored off-chain on the Cloud File Storage.

We assign an application identifier to each application. The identifier should be able to uniquely identify an application. For example, `ir.ac.modares.myapp` can identify `myapp` among the apps developed by the domain holder of `modares.ac.ir`. The application developers also choose a user-friendly title for their application which is shown in the UI and may be non-unique.

Each developer in our proposed app store is identified by his/her blockchain account. All accounts correspond to asymmetric signing key pairs. Each application package in our proposed app store is digitally signed by the private key of its developer. All the information about the identity of the developer is stored and managed in a the decentralized way by the Ether-DID mechanism we explained before. Only the controller of the identity or one of its delegates is allowed to add or update application-related information or publish packages.

It is also possible for an application identifier to belong to multiple developers. This feature opens the possibility for the developers to change their application signing keys. Some platforms including the Android platform do not allow the installation of an update that is signed by a key different from the one used to sign the already installed package. In this case, the application package can be signed by new keys for use by new installations, and the old keys can be used for updates to existing installations. While these applications share the same identifier, they are treated differently by our smart contracts because their developers are different. Therefore, support for applications with multiple signing keys may be implemented in the UI. A better way to do this is to give a choice to the user to investigate the credentials of both developers to better protect them against fraud.

Application licenses in our proposed app store are treated as transferrable *non-fungible tokens* (NFT), and the app store smart contract implements all the functionality required to issue and transfer these tokens, as well as to keep track of their ownership. Each license token has a unique $tokenId$, and the combination of application and developer identifiers is stored in its $tokenUri$ to reflect its correspondence to a specific application. The results of all operations are recorded in the contract's internal blockchain storage, can be queried and verified securely, and are inherently immutable outside the control of the contract.

In the following subsections, we first explain how applications are registered and published. Then, the operations and issues related to license management are discussed.

4.3.1 Application Registration

Before releasing an application in our proposed app store, its developer shall register the application, its price, and his/her blockchain address for receiving payments. The package price can be specified in the unit of any on-chain token (e.g., an ERC-20 token). The token's contract address is also provided along with the price. We decided not to rely on the native cryptocurrency of the blockchain because cryptocurrencies are too volatile, and the users and developers may wish to use a token (e.g., a stable coin that is pegged to fiat money).

Application registration is accomplished by invoking the *register* method of the app store smart contract in a blockchain transaction. The blockchain platform verifies the identity of the caller (transaction signature) and passes this identity as a built-in variable to the contract. The app store ensures the caller is the controller of the developer identifier or one of its delegates, and that the combination of the application and developer identifiers is not registered before, and then, registers the application information in its internal storage on the blockchain.

4.3.2 Application Publication

Once a new version of an application is ready, different platform-specific packages can be published. Each specific build has a different download URL and digital signature and can be installed on specific operating system versions. All the packages distributed in a single release share the same version name which is shown to the users in the UI. However, each of them has a different version code.

More specifically, we identify each build by a version code which is a unique integer number not shown to the user. The version code is internally used to distinguish between different builds of an application, and select the most-recent compatible build for the user.

In each release of an application, the developer builds one or several packages, digitally signs each of them separately by the private key corresponding to the application's developer identifier, and uploads them to the FS. Then, invokes the *publish* method of the app store smart contract in a blockchain transaction. The contract verifies that the application has been registered before and that the caller is the controller (or a delegate) of the developer identifier. If a particular version has not been published before, the contract checks the package signature to match the public key of the developer identifier. If all these checks succeed, the package information will be stored in the internal storage of the contract on the blockchain.

4.3.3 Search Functionality

The app store smart contract implements several inquiry methods to get the information related to an application according to its identifier, a combination of application and developer identifiers, and details about a specific version code of an application. The app store contract also *logs* all the relevant information of application registrations and publications on the blockchain. These logs are recorded in the blockchain and are accessible to external entities (off-chain). The UI uses these logs to discover all details about every application in order to support search functionality other than the before mentioned exact matches.

4.3.4 License Purchase

Our license tokens are *minted* by the app store upon purchase of a license. In particular, once a user decides to purchase an application license, the user must first approve the transfer of at least the required amount of the appropriate token by the app store contract. Then, invokes the *purchase* method of the app store contract in a blockchain transaction by specifying the requested application and the new license owner address. The method retrieves the deposit address of the application developer, and its price information. Then, directly transfers the price of the license from the account of the user to the account of the developer. If the transfer is successful, it mints a fresh license token for the address specified by the user and sets the $tokenUri$ to the combination of application and developer identifiers.

4.4 License Validation Process

In our proposal, each application installation is identified by an asymmetric signing key pair. The key pair shall be securely generated by the runtime platform and be only accessible to the application. We denote the corresponding public and private keys by PK and PR, respectively. The PR shall be stored in secure storage and be protected from direct access by the application or from extraction or backup [12].

4.4.1 Activation

All applications which use our license validation process, generate a fresh key pair upon the first run. All users can download an application, but if it requires a license, at the start of running the application, the

[12] For example, the Android platform supports a *KeyStore* by API levels 23 and onwards (Android 6+). The *KeyStore* can be used to generate ECDSA key pairs which are `ORIGIN_GENERATED` and not extractable. There are API calls to sign or decrypt messages by a private key without the key ever exiting the *KeyStore*.

user shall activate the license to be used for the specific installation of the application. The activation is as simple as transferring a purchased license token from the address of the user to the address corresponding to PK (e.g., scanned from a QR-code shown by the application at startup.) Once the transfer is successful, the user enters the $tokenId$ in the application. The application checks the $tokenUri$ has the correct application and developer identifiers, and the ownership of the token is recorded in the app store. The result of this check can be cached on tamper-proof storage on the runtime platform, or be repeated before performing critical tasks.

4.4.2 License Transfer

License transfers are also possible in our proposal. The process starts with an already-activated installation. First of all, the application invalidates the cached information about activation. Then, contacts the app store contract to transfer ownership of its license token to the address of its user (determined, e.g., by scanning the QR code of the receiving address.) Now, the installation is deactivated and the user can follow the activation process to activate another installation. It is also possible to hand over the license to other users.

4.4.3 Server-Side License Validation

If an application relies on server-side services, the PK can also be used to uniquely identify the installation in a pseudonymous fashion. The server should expect some proof to ensure the application has the corresponding PR (e.g., a digital signature on the requests). The server can easily verify that PK owns a valid license token from the app store contract, and provide the requested service accordingly.

4.5 Discussion and Limitations

In this section, we have a complimentary discussion about our proposal. Smart contracts have only access to the blockchain and other smart contracts. Therefore, the *publish* method of the app store cannot connect to the Internet and match the package signature with the uploaded file. If the FS supports content addressing (e.g., IPFS), the contract can also match the content address with the package signature. In other words, if the package URL refers to the hash of the uploaded file for accessing it, the contract can provide a stronger assurance about the authenticity of the uploaded package files.

The integrity and authenticity of the application code are essential for the license validation process. If a manipulated version of the application is installed, the whole license validation process can be circumvented. This fact is not specific to our proposal, but as we mentioned in the background section of the paper, all software license validation mechanisms should be protected by properly obfuscating the application code, and preventing any modifications in the code. Our proposed app store enables everyone including the application to retrieve the signature of the authentic application (for developer identifier) and to match it with the signature of the installed package. The application should terminate in case it discovers that the installation package was inauthentic.

We do not link the real identity of the users to the licenses at all. In other words, any user with a blockchain wallet can use our proposed app store without registering any *personally identifiable information* (PII). In addition, every application which is installed on a device uses a different installation key pair than the other applications on that device. Therefore, one cannot use PK to identify a device or link data from different applications on a device together. However, other background information such as the source IP address of connections or the information which are collected by the application may be used to identify a device or its user. In these cases, the user should use anonymity tools for his privacy.

There is a corner-side misuse risk for license tokens that cannot be prevented without using trusted hardware on the client side. In particular, if a server-side service is intended to be only provided to license holders, there is no way for the server to reliably verify which application is using a license token. The token may be presented to the server by an original application or by another application that is imitating the behavior of the original application for the server. Note that while the application may be in-authentic, our license validation cannot be circumvented and a valid license token must be used for the requests. However, if the services are personalized or require purchases other than the license, the mentioned malicious user cannot gain any advantage by using a license outside of an authentic application. By the way, we advise the service providers to have a service quota for each PK to minimize this type of risk.

5 Evaluation

We have implemented a proof-of-concept app store contract, and base our performance evaluations on it. In addition, we compare our proposal with the related works, according to their license management features.

5.1 Implementation

Our proof-of-concept code is implemented in Solidity with OpenZeppelin [13] library, and deployed on the Ropsten (Ethereum) and RSK testnets. In particular:

- We implemented our proposed app store contract by extending OpenZeppelin's `ERC721URIStorage` (in Solidity). The license token complies with ERC-721 [29].
- We call upon the EtherDID registry [14] contract for the identification of the developers. The contract was already deployed on the Ropsten and RSK test networks.
- In addition, we deployed a basic ERC-20 token to be used as the on-chain token in purchases. The token was basically the OpenZeppelin's `ERC20` contract, instantiated with a custom name and initial supply.
- We prepared some test data and called different app store contract methods in NodeJS (JavaScript) and by calling appropriate TruffleSuite's `Web3js` methods [14]. Our performance evaluations were based on this code.
- We connected our code to the Ropsten network through the Infura cloud, and to the RSK test network by a network-provided public node [15].

We ran our proof-of-concept code on Ethereum and RSK testnets. Then, measured the exact costs in terms of gas consumption. As the exact inter-block time and gas price are dependent on the real-time dynamics of the blockchain, we use the results of these measurements, and estimated the real-time cost and latency of our proposal regarding the real-time Ethereum and the RSK mainnet statistics on a specific date (i.e., February 1, 2022). The results of our evaluations are summarized in Table 1, and discussed in the subsequent subsections.

5.2 Latency

The latency of our proposed app store is mainly related to confirmation times for Ethereum transactions. The number of transactions required to perform each of the basic functionalities in our proposed app store are demonstrated in Table 1. In particular, searching, downloading, and license verification do not require writing to the blockchain, and do not have a significant latency. However, application registration, publication, license verification, and license transfer each require one transaction. Only the purchase functionality requires two transactions. This is because on-chain tokens must be approved to be transferred from the user's address in one transaction, and the actual purchase is performed in a subsequent transaction. If the user trusts in the app store contract, it can be approved for transferring unlimited amount of tokens. In this case, subsequent purchases can be performed in one transaction. Accordingly, the latencies of the functionalities based on real-time statistics of the Ethereum and RSK mainnets are estimated in Table 1.

5.3 Costs

The main costs of our proposed app store for its founders are:

- the development cost,
- the deployment cost for the app store contract (deployment transaction fee),
- a web server to host the server side of the UI, and
- a database for caching applications' metadata.

A commercial implementation should optimize its smart contract code to minimize gas usage because the gas fee is determined according to the rules of the blockchain, and is out of our control. However, we believe that by letting the developers host their package files on third-party cloud file storage,developers and users do not pay subscription we minimized server-side requirements, and the costs other than the blockchain fees are minimal.

The developers and users do not pay subscription fees or alike to like the app store. In addition, license payments are directly transferred to the developers' accounts. Nonetheless, all functionalities which require issuing a transaction and writing some data on the blockchain, incur a gas fee to the initiator of the transaction (user or developer).

The mentioned blockchain gas costs are highly dependent on the implementation, the dynamics of the blockchain, and the price of the native cryptocurrency in exchanges (gas price). Our proof-of-concept implementation is not optimized for gas cost and only incorporates the essential functionalities. We deployed our implementation on the Ethereum Ropsten and RSK testnets which are quite similar to the mainnets.

Therefore, our gas measurements should give an insight into the real costs.

According to our estimates in Table 1, the most costly operation on both blockchains is the deployment of the contract. Then, publication, purchase, and application registration come in descending order. Finally, the license activation/transfer is the cheapest operation. We used dynamic arrays in our implementation, and therefore, the gas cost of the method calls may slightly differ according to the history of calls.

[13] `https://openzeppelin.com/`
[14] `https://www.trufflesuite.com/`
[15] `https://public-node.testnet.rsk.co`

Table 1. Latencies and Costs of the App Store Functionalities.

Functionality	Smart Contract Method	Tx's	Ethereum Blockchain			RSK Blockchain		
			Latency[a] (sec's)	Gas (max)	Cost[b] (USD)	Latency[a] (sec's)	Gas (max)	Cost[c] (USD)
Contract deployment	-	1	13.19	3 645 632	810.25	31.53	4 448 092	11.05
App registration	*register*	1	13.19	169 358	37.64	31.53	157 238	0.39
Publication	*publish*	1	13.19	270 492	60.12	31.53	259 340	0.64
Download	*getAppVersion*	0	-	-	free	-	-	free
Search	- (log access)	0	-	-	free	-	-	free
	getAppVersion	0	-	-	free	-	-	free
Purchase	*approve* (on-chain token)	1	13.19	46 821	10.41	31.53	46 021	0.11
	purchase	1	13.19	204 580	45.47	31.53	177 756	0.44
Activation	*transfer*	1	13.19	58 652	13.04	31.53	52 692	0.13
License verification	*ownerOf*	0	-	-	free	-	-	free
License transfer	*transfer*	1	13.19	58 652	13.04	31.53	52 692	0.13

[a] According to the average block time on Feb. 1, 2022.
[b] According to the average gas and Ether prices on Feb. 1, 2022 (81 Gwei and 2 743.87 USD, respectively).
[c] According to the average gas and RSK prices on Feb. 1, 2022 (0.065164 Gwei and 38 139.28 USD, respectively).
References for all the mentioned stat's: `https://coinmarketcap.com/`, `https://etherscan.io/`, and `https://stats.rsk.co/` (Accessed on Feb. 1, 2022).

Hence, we reported the maximum gas cost that we recorded during our tests.

While the Ethereum and RSK virtual machines are quite similar, and we deployed the same code on both networks, the amount of gas needed for each method call differs in the two blockchains. All functionalities consume less gas in the RSK blockchain than the Ethereum blockchain. In contrast, contract deployment in the former consumes 22.01% more gas than the latter. It is worth noting that gas is much cheaper in RSK than in Ethereum. Therefore, the estimated fee to perform the tasks are cheap (less than 1.00 USD) on the RSK in contrast to the very expensive fees on the Ethereum.

5.4 Comparison with the Related Works

In this section, we compare our proposal to Google Play Store [30] and other related works concerning their software license management. We should note that the work presented in [20] is too general to be considered a specific license management solution and be comparable to the other more-concrete works. A summary of the comparison is presented in Table 2, and we explain the compared features in what follows.

With respect to the *type of blockchain*, the works based on private blockchains [9, 22, 24] can enjoy fast and free transactions. However, they are too costly from the governance point of view. Either a multi-company (consortium) blockchain should be initiated for this purpose, or the solution cannot bring about the critical features of a blockchain for license management. Access to private blockchains is restricted to designated entities and one can question their transparency and integrity guarantees. In addition, they are not decentralized and are under the control of their stakeholders. In this regard, they are too costly in comparison with a centralized cloud-based solution.

Regarding the *autonomy* aspect, our proposal is fully autonomous and is operated by smart contracts. Nothing is needed to intervene to accomplish tasks, other than what blockchain offers. The operations of these smart contracts are fully auditable and transparent to the public. Once a transaction is completed and confirmed by the blockchain, it is irreversible and cannot be manipulated by anyone. Other proposals which do not depend on smart contracts [9, 21, 22, 30] are not autonomous. The works presented in [26, 27] are semi-autonomous because they use off-chain methods for encrypting secret license information (IPFS hash and bootstrap code, respectively).

In consideration of the *developer authentication*, we noticed that there is some gap between identities in a blockchain and real-world identities. In addition, blockchain smart contracts cannot directly contact the Internet. This is a challenging limitation for an app store. Because the authenticity and originality of

Table 2. Comparison with the Related Works.

	Type of blockchain	Autonomous	Developer authentication	License validation*	License transfer**	User On-Boarding
Google Play Store [30]	-	false	off-chain	user (online)	false	Easy
[21]	public (Namecoin)	false	off-chain	user (online)	false (inefficient)	Hard
[9, 22]	private	false	off-chain	user (online)	false (inefficient)	Hard
[23]	public (Ethereum)	true	false (optional email)	user (online)	false (efficient)	Hard
[24]	private (Ethereum)	true	off-chain	device (online)	false (efficient)	Hard
[25]	public (Ethereum)	true	off-chain	device (continuous online)	false (efficient)	Hard
[26]	public (Ethereum)	semi-autonomous	false (reputation system)	false	not applicable	Hard
[27]	public (Ethereum)	semi-autonomous	false	user (online)	false (inefficient)	Hard
This proposal	public (Ethereum, RSK)	true	SSI	device (one-time online activation)	true	Hard

* *User:* The license is issued to the users and can be used on all their devices. *Device:* The license is bound to a specific device. *Online:* An Internet connection is required to validate the license, but after that, the user may disconnect and continue using the application. *Continuous online:* Internet connection is required to validate the license. If the user is disconnected from the Internet, the application is not usable. *One-time online activation:* Internet connection is only required to activate a license, and future validations do not need an Internet connection.

** *Inefficient:* not proposed but can be implemented in a costly manner. *Efficient:* not proposed but can be implemented efficiently. *Not applicable:* No mechanism is proposed to stop users from sharing copies of the application between users.

an application are crucial for users. In the worst case, a malicious person may register another developer's application in the app store, and sell fake licenses to earn money. In addition, malware can easily spread in this case. Therefore, the anonymity of developers is not a desirable feature for an app store. Most solutions rely on off-chain authentication mechanisms [9, 21, 22, 24, 25, 30]. Developers using Spheris [23] or [26] can keep their anonymity, and a reputation and feedback system is provisioned to boycott malicious applications and developers. This feature is provided in some of the other app stores. However, none of the proposals use attack-resistant reputation methods, and biased feedback can negatively affect this system.

We authenticate the developers using their on-chain EtherDIDs. Our smart contracts ensure that only the controller (or a delegate) of a DID (developer) may work with our system. In addition, the contracts verify the signature of the developer before publishing application packages. The application should also verify its signature by the one registered in our proposed app store before activation. The information backing the mentioned procedure is securely stored on the blockchain. However, we cannot be sure that the intellectual property belongs to the developer who registers an application in our proposed app store. As our system is fully autonomous, we cannot prevent someone from publishing in-authentic applications. However, we bind the applications to the identity of their developers, and even the same application identifier can belong to multiple developers.

Therefore, on the one hand, a malicious developer cannot prevent another developer from publishing the the original version of an application. On the other hand, we support DIDs which are a foundation for *Verifiable Credentials* (VCs), and our reliance on such a foundation opens the door to the realization of this promising framework. Thus, the users and the developers can utilize the full potential of DIDs by exchanging and using VCs for identification and trust development. Upon this, the users can build trust in the identity of the developers in a self-sovereign way, and check the credentials of the developers before trusting their published application.

According to our review of the close related works, *license validation* in most of the related works [9, 21–23, 27, 30], is performed for individual users and are not confined to specific devices. In addition, license validation is based on online information [9, 21–25, 27,

30]. While it is possible and convenient for the user to cache license information on the device, there is no guarantee that the license is not being used for another device, at the same time. Our proposal limits the use of a license to one device which has the installation private key. The key is never transferred over the Internet or even allowed to exit the secure storage of the device. Therefore, in our proposal, the device is only required to connect to the Internet during the activation time.

We feature an easy *license transfer* method. Other related works do not offer this feature and require each user to purchase a fresh new license. In some cases, license transfer is inefficient for implementation. In [21] the license token must be traced back to the account of the developer, which is inefficient in UTXO-based blockchains, and impossible if performed carelessly. In [9, 22, 27], the bootstrap code is encrypted by the license holder's private key, and this code should be re-encrypted with the new holder's key in an off-chain process. However, this feature can be implemented easily in some other works [23–25, 30]. We believe this feature is highly desirable in the device-bound licenses which we and [24, 25, 27] offer. This allows users to, for example, re-install their operating system without worrying about losing their license.

If we consider the *user onboarding* aspect, a traditional cloud-based app store, like the Google Play Store, is the easiest and most familiar method for users. All blockchain-based proposals require the users to be familiar with the concepts of cryptocurrency and use a blockchain wallet, and our proposal is not an exception. If a user loses access to the wallet, or the private key is deleted from it, there is no way to recover the key. On one hand, it is related to the strong security of blockchain accounts that cannot be accessed by anyone other than their owners. On the other hand, it is not that user-friendly. Another consequence related to the use of blockchain is that the users should own cryptocurrency to pay for transaction fees and license purchases. These aspects make our proposal along with other blockchain-based proposals more complex for users who are not familiar with blockchain concepts.

6 Conclusions

We introduce an autonomous decentralized app store in this paper which is backed by a public blockchain and smart contracts. The app store does not rely on any intermediaries, and/or modifications to the blockchain or its transactions. It can be used by everyone, ranging from independent developers to large software vendors. All the core app store functionalities are implemented by smart contracts and operate autonomously. Therefore, in addition to being distributed, our proposed app store does not have centralized management. We feature a license validation solution for our proposed app store which relies on a public blockchain, is fully autonomous and authenticates the developers by their decentralized identifiers and self-sovereign identities. Licenses in our proposal are bound to the user's device and can be transferred securely to other devices. Transaction fees and latency are the main drawbacks of our proposed app store and the other solutions which rely on public blockchains.

References

[1] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman. A Survey of App Store Analysis for Software Engineering. *IEEE Transactions on Software Engineering*, 43(9):817 – 847, 2017. ISSN 0098-5589. doi:10.1109/TSE.2016.2630689.

[2] T. Erdbrink and V. Goel. Apple, Citing U.S. Sanctions, Removes Popular Apps in Iran. `https://www.nytimes.com/2017/08/24/technology/apple-iran.html`, Date Accessed: Sep. 2, 2021.

[3] J. Li, A. Grintsvayg, J. Kauffman, and C. Fleming. LBRY: A Blockchain-Based Decentralized Digital Content Marketplace. In *2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pages 42–51. IEEE, 2020. ISBN 978-1-7281-6978-1. doi:10.1109/DAPPS49028.2020.00005.

[4] J. Crussell, C. Gibler, and H. Chen. Attack of the Clones: Detecting Cloned Applications on Android Markets. In *European Symposium on Research in Computer Security*, page 37–54. Springer, 2012. ISBN 978-3-642-33166-4. doi:10.1007/978-3-642-33167-1_3.

[5] J. Crussell, C. Gibler, and H. Chen. AnDarwin: Scalable Detection of Semantically Similar Android Applications. In *European Symposium on Research in Computer Security*, page 182–199. Springer, 2013. ISBN 978-3-642-40202-9. doi:10.1007/978-3-642-40203-6_11.

[6] A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz. On blockchain and its integration with IoT. Challenges and opportunities. *Future Generation Computer Systems*, 88:173–190, 2018. doi:10.1016/j.future.2018.05.046.

[7] S. Pech. Copyright Unchained: How Blockchain Technology Can Change the Administration and Distribution of Copyright Protected Works. *Nw. J. Tech. & Intell. Prop.*, 18, 2020.

[8] K. Thompson. Reflections on Trusting Trust. *Communications of the ACM*, 27(8):761–763, 1984.

[9] A. Litchfield and J. Herbert. ReSOLV: Applying Cryptocurrency Blockchain Methods to Enable Global Cross-Platform Software License Validation. *Cryptography*, 2(2), 2018. ISSN 2410-387X. doi:10.3390/cryptography2020010.

[10] X. Xu, I. Weber, and M. Staples. *Architecture for Blockchain Applications*. Springer, 2019.

[11] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, 2016.

[12] D. Reed, M. Sporny, D. Longley, C. Allen, R. Grant, M. Sabadello, and J. Holt. Decentralized Identifiers (Dids) V1. 0: Core Architecture Data Model and Representations. 2020. `DecentralizedIdentifiers(DIDs)v1.0(w3.org)`, Date Accessed: May. 10, 2021.

[13] N. Mohammadzadeh, S. Dorri Nogoorani, and J. Muñoz-Tapia. Decentralized Factoring for Self-Sovereign Identities. *Electronics*, 10(12), 2021. ISSN 2079-9292. doi:10.3390/electronics10121467.

[14] P. Braendgaard and T. Joel. EIP-1056: Ethereum Lightweight Identity. `https://eips.ethereum.org/EIPS/eip-1056`, Date Accessed: Sep. 2, 2021.

[15] J. Benet. IPFS - Content Addressed, Versioned, P2P File System. *arXiv preprint arXiv:1407.3561*, 2014.

[16] G. S. Mendes, D. Chen, B. M. Silva, C. Serrao, and J. Casal. A Novel Reputation System for Mobile App Stores Using Blockchain. *Computer*, 54(2):39 – 49, 2021. ISSN 0018-9162. doi:10.1109/MC.2020.3016205.

[17] C. Cassano. Deco.Network Alpha Smart Contracts. `https:/medium.com/deconet/e6ac31f7ef35`, Date Accessed: June. 7, 2022.

[18] B. Ahmad, J. Dujaka, E. Herwin, and N. Sauer. Serverless Software License. *Project Report*, 2020.

[19] D. Tkachenko. How to Develop an Ethereum Smart Contract for Licensing? `https://www.apriorit.com/dev-blog/557-ethereum-smart-contract-licensing`, Date Accessed: June. 7, 2022.

[20] V. Stepanova and I. Eriņš. Blockchain-Based Model for Software Licensing. In *2019 4th International Conference on System Reliability and Safety (ICSRS)*, pages 30–34. IEEE, 2019. ISBN 978-1-7281-4782-6. doi:10.1109/ICSRS48664.2019.8987715.

[21] C. Fortin. Master Bitcoin - The Proof of Ownership. Technical Report, 2011. `https://frozenlock.files.wordpress.com/2011/11/master-bitcoin.pdf`.

[22] J. Herbert and A. Litchfield. A Novel Method for Decentralised Peer-to-peer Software License Validation Using Cryptocurrency Blockchain Technology. In *Proceedings of the 38th Australasian computer science conference (ACSC 2015)*. AUT, 2015.

[23] D. Shabun. Spheris Whitepaper. Technical Report, 2017. `https://spheris.io/spheris_whitepaper.pdf`.

[24] A. Seitz, D. Henze, D. Miehle, B. Bruegge, J. Nickles, and M. Sauer. Fog Computing as Enabler for Blockchain-Based IIoT App Marketplaces - A Case Study. In *2018 Fifth International Conference on Internet of Things: Systems, Management and Security*, pages 182–188. IEEE, 2018. ISBN 978-1-5386-9586-9. doi:10.1109/IoTSMS.2018.8554484.

[25] F. Magnanini, L. Ferretti, and M. Colajanni. Efficient License Management Based on Smart Contracts Between Software Vendors and Service Providers. In *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*, pages 1–6. IEEE, 2019. ISBN 978-1-7281-2523-7. doi:10.1109/NCA.2019.8935038.

[26] J. Park, S. Lee, G. Kim, and J. Ryou. Decentralized Blockchain-Based Android App Store with P2P File System. In *Advances in Computer Science and Ubiquitous Computing*, page 525–532. Springer, 2018. ISBN 978-981-13-9340-2. doi:10.1007/978-981-13-9341-9_90.

[27] M. Moharrer Monem and S. D. Nogoorani. A Decentralized App Store Using the Blockchain Technology. In *2020 17th International ISC Conference on Information Security and Cryptology (ISCISC)*, pages 14–21. IEEE, 2018. ISBN 978-1-7281-8570-5. doi:10.1109/ISCISC51277.2020.9261911.

[28] A. P. Kryukov and A. Demichev. Decentralized Data Storages: Technologies of Construction. *Programming and Computer Software*, 44(5): 303–315, 2018. doi:10.1134/S0361768818050067.

[29] W. Entriken and D. Shirley and J. Evans and N. Sachs. EIP-721: Non-Fungible Token Standard. Technical Report, 2021. `https://eips.ethereum.org/EIPS/eip-721`.

[30] Android Developers Documentation. App Licensing. `https://developer.android.com/google/play/licensing`, Date Accessed: Aug. 17, 2021.

Sadegh Dorri Nogoorani is an assistant professor and the director of the Blockchain Laboratory at Electrical and Computer Engineering Faculty of Tarbiat Modares University. He received his B.S. degree in Software Engineering from the University of Tehran, and his M.S. and Ph.D. degrees respectively in Computer Networks and Software Engineering both from Sharif University of Technology. His research focus is on the blockchain and distributed ledger technology and is also interested in security, privacy, and trust in distributed systems (cloud, P2P, grid).