energies

Article

High-Secured Data Communication for Cloud Enabled Secure Docker Image Sharing Technique Using Blockchain-Based Homomorphic Encryption

Vishnu Kumar Kaliappan [1], Seungjin Yu [2], Rajasoundaran Soundararajan [3], Sangwoo Jeon [2], Dugki Min [2,*] and Eunmi Choi [4]

1 Konkuk Aerospace Design-Airworthiness Research Institute (KADA), Konkuk University, Seoul 05029, Korea; vishnudms@gmail.com
2 Department of Computer Science and Engineering, Konkuk University, Seoul 05029, Korea; seunggin.yu@gmail.com (S.Y.); ndrw5580@gmail.com (S.J.)
3 School of Computing Science and Engineering, VIT Bhopal University, Bhopal 466114, India; rajasoundaransraja@gmail.com
4 Department of Computer Science and Engineering, Kookmin University, Seoul 05029, Korea; emchoi@kookmin.ac.kr
* Correspondence: dkmin@konkuk.ac.kr

Citation: Kaliappan, V.K.; Yu, S.; Soundararajan, R.; Jeon, S.; Min, D.; Choi, E. High-Secured Data Communication for Cloud Enabled Secure Docker Image Sharing Technique Using Blockchain-Based Homomorphic Encryption. *Energies* **2022**, *15*, 5544. https://doi.org/10.3390/en15155544

Academic Editors: R. Maheswar, M. Kathirvelu and K. Mohanasundaram

Received: 23 June 2022
Accepted: 26 July 2022
Published: 30 July 2022

Abstract: In recent years, container-based virtualization technology for edge and cloud computing has advanced dramatically. Virtualization solutions based on Docker Containers provide a more lightweight and efficient virtual environment for Edge and cloud-based applications. Because their use is growing on its own and is still in its early phases, these technologies will face a slew of security issues. Vulnerabilities and malware in Docker container images are two serious security concerns. The risk of privilege escalation is increased because Docker containers share the Linux kernel. This study presents a distributed system framework called Safe Docker Image Sharing with Homomorphic Encryption and Blockchain (SeDIS-HEB). Through homomorphic encryption, authentication, and access management, SeDIS-HEB provides secure docker image sharing. The SeDIS-HEB framework prioritizes the following three major functions: (1) secure docker image upload, (2) secure docker image sharing, and (3) secure docker image download. The proposed framework was evaluated using the InterPlanetary File System (IPFS). Secure Docker images were uploaded using IPFS, preventing unauthorized users from accessing the data contained within the secure Docker images. The SeDIS-HEB results were transparent and ensured the quality of blockchain data access control authentication, docker image metadata denial-of-service protection, and docker image availability.

Keywords: cloud computing; secure communication; docker security; homomorphic encryption; virtualization; blockchain; docker image sharing

1. Introduction

Over the past decade, a boom in virtualization technologies has made it possible to divide a computer system into many discrete virtual environments. Technology has progressed quickly and offers many advantages. The virtualization of servers in data centers is one of the most popular uses of virtualization technology. An administrator can set up one or more virtual system instances on a single server using server virtualization. These virtual servers may be hired on a subscription basis and function similarly to physical servers. The growing adoption of virtualization technologies drives the desire for a virtualization solution that can offer dense, scalable, and secure user environments. There are various virtualization products on the market right now. They can be divided into the following two primary categories: container-based virtualization and hypervisor-based virtualization. Container-based virtualization is also referred to as operating system virtualization since it allows the virtualization layer to run as an application along with the operating systems [1].

Containers emerged as mini virtual computers, which are lighter and more efficient because a single operating system can manage all hardware. This means that ten times more virtual environments can be run on a physical server than in hypervisor-based virtualization [2]. Containers are considered to be the future of virtual machines [3].

It is typical practice to use docker containers for provision across a shared physical host. A docker image repository simplifies the publishing and shares high-quality docker images generated by docker. Although the rapid deployment and sharing of docker images have the significant advantage of allowing developers to share a wide range of real-time apps [4,5], because there is no specific mechanism for dealing with docker image vulnerabilities, the docker platform is easily vulnerable to numerous security threats. The older packages containing vulnerabilities in the Docker image can be vulnerable to attacks such as DoS, acquire privilege, and so on [6,7].

The distribution of bitcoin mining software is one example of an attack using Docker images; this attack demonstrates that a variety of attacks may be carried out using Docker images irrespective of the target Operating System (OS). As a result, one of the critical difficulties in establishing a reliable Docker environment is the security of Docker images. The fundamental cause of this Docker image vulnerability issue is users failing to perform a separate security verification task on downloaded docker images or docker images that are about to be submitted to a docker image repository.

In light of the preceding factors, we proposed a secure docker image sharing framework with homomorphic encryption and blockchain (SeDIS-HEB) in this paper to secure the docker images. The proposed framework uses homomorphic encryption to offer authentication and access control to metadata for secure docker image sharing. During the uploading step, the SeDIS-HEB framework scans the docker image, extracts the CBE list of the docker, encrypts the matching docker image, and uploads it. Each value is called ImageBCID and is based on hash values, symmetric keys, and user information. The image is passed through user authentication during the sharing procedure via ImageBCID. Later, the system was designed so that it could only be imported via ImageBCID authentication by encrypting and recording the metadata to obtain docker images and CVE lists in the blockchain. The uploader proceeds with the sharing procedure using the ImageBCID issued in the subsequent sharing process. The uploader initially authenticates the user with authorization to the related docker image using the issued ImageBCID and personal information. It then provides access to docker metadata on top of the blockchain for users who pass the authentication. Finally, it generates a new ImageBCID by fusing metadata and homomorphically encrypted data from users that the uploader wishes to share with. The shared user receives the shared docker image via their personal information and ImageBCID.

This study makes the following significant contributions.

- A new paradigm for improving security in docker image sharing is proposed. The Docker Image Sharing with Homomorphic Encryption and Blockchain (SeDIS-HEB) platform enables metadata authentication and access control. The following three key features were developed to ensure authentication and access control: secure docker image upload, secure docker image sharing, and secure docker image download.
- The system structure for secure docker image sharing was implemented for the docker image, ensuring integrity via IFPS.
- The proposed framework was validated with the Proof of concept.

The remainder of the paper is organized as follows: Section 2 elaborates on the background and current research findings of security measures for docker images. Section 3 describes the potential threads in docker images. Section 4 presents the proposed Secure Docker Image Sharing with Homomorphic Encryption and Blockchain (SeDIS-HEB) framework. Section 5 presents the implementation and results analysis, and Section 6 explores the conclusion and future directions.

2. Background

2.1. Docker Image Systems

Docker image—The combination of program source code and library required for service operation is called a Docker image [8]. The service is launched by placing the docker image on top of the docker image container. Docker images must be chosen as the default option, and Docker base images are read-only files. The base image is typically comprised of operating system docker images such as Ubuntu and Debian. Users can then use different environments by layering the packages they want to use on top of the corresponding base image. Docker images can be built using docker build. Docker images can be built in the Advanced Union Mount File System format (AUFS). The AUFS file system supports the mounting of multiple image files using containers. Docker images save much space on these systems by reusing the images used to develop other concepts, allowing for a faster generation.

InterPlanetary File System (IPFS)—Docker images are usually heavy due to their nature. Hence, storage for them in the blockchain is limited. It takes longer at the start to save data in the blockchain since it mines blocks first and then saves data in them. Furthermore, while storing data in blocks, there is a problem with expanding the size of the block. We used IPFS to efficiently separate data storage [9]. A distributed P2P file system that connects all computers is known as an interplanetary File System (IPFS). When the files are shared, IPFS assist in partitioning and distributing them to nodes in the IPFS network. The IPFS hash is returned to the user based on the uploaded file's contents, which will be recognized as the same file if it is kept on the network under a different name. The distributed file can be accessed using the IPFS hash when a user requires a file.

2.2. Secure Docker Image Systems

CVE (Common Vulnerabilities and Exposures)—CVE is an abbreviation for a publicly available list of computer security issues. It is typically referred to as the CVE ID number allocated to a security problem. CVE allows professionals to collaborate to prioritize and address vulnerabilities to manage safer computer systems. CVE is overseen by MITRE Corporation [10], which receives funding from the US Department of Homeland Security's Cybersecurity and Infrastructure Security Agency. When a new CVE is reported, the CVE database registers and identifies each new CVE, allowing developers to stay up to date with the latest information by locating and continuously updating the system contained in their approach. CVE also provides a CVSS score [11] for security, allowing users to determine how secure the data is.

Docker Scan—Docker scanning can generate a list of CVEs for docker images. First and foremost, because the docker image is composed of layers, as previously mentioned, the docker scan examines the packages in each layer. Once verification is completed, the NVD dataset [11] is checked for security issues. In this procedure, we cross-check the CVE list and return the stated information to the user. Anchore [12], Clair [13], and Snyk Engine [14] are examples of standard Docker image scanning software. In this work, we utilized Snyk Engine docker scans to scan images that users download from the hub or their docker images.

Blockchain—Satoshi Nakamoto invented blockchain, a distributed book system, in 2008 [15]. Blockchain technology is primarily kept in a chain-based distributed storage system based on peer-to-peer (P2P) networks of small data units known as "blocks". It is a distributed book storage format that allows arbitrary users to modify the blockchain and anyone to inspect transactions or modifications through the blockchain. This research proposes a distributed system that stores metadata using blockchain as a database via a distributed book storage structure [16]. Each node must be mined to add data to the blockchain. A consensus technique known as proof-of-work [17] is required for each node in the mining process for each new block.

Furthermore, denaturing the data necessitates modification of the majority of the node's books, which is difficult. Blockchain was previously made available to anybody via public blockchain but currently employs private blockchain for usage by a specific organization or hybrid blockchain for a combination of the two. In this research, we used the Ethereum public blockchain.

Smart contract—Ethereum is a public blockchain that may be used to build decentralized applications [18]. The smart contract is a characteristic that distinguishes the Ethereum blockchain from others. An intelligent contract [19], a notion of Ethereum founder Vitalik Buterin, outlines the process that must be executed in advance, and when requested by the contract, it acts as described in the smart contract. The benefit of this method is that if the requester meets the contract requirements without the assistance of a third-party intermediary, the requester provides the desired data and returns it, and if this does not satisfy the criteria, the contract is stopped. Smart contracts are written in their own languages, which can be written in Solidity [20], which is similar to object-oriented programming languages such as JAVA and C++.

Symmetric, asymmetric encryption—In this work, we used various encryption methods to ensure the confidentiality of the data. The encryption method employs symmetric, asymmetric, and homomorphic keys. "Symmetric key encryption" refers to algorithms that enable encryption and decryption using the same key. Symmetric encryption offers the advantage of swiftly encrypting any data. This paper proposed an encryption procedure for datasets containing a large amount of data. The AES256 technique generates a key and encrypts the data using the generated key.

In asymmetric encryption [21], the user has a public and a private key, unlike symmetric keys. Data encrypted with a public key can be decrypted with a private key and vice versa. Asymmetric encryption has the advantage that symmetrically encrypted keys can only be utilized by users who encrypted keys with public keys for that symmetric encryption. Furthermore, asymmetric encryption is used for data registered in the blockchain to maintain the confidentiality of the data transmitted.

Homomorphic Encryption—Homomorphic encryption is a cryptographic method designed to allow operations to be carried out with encrypted data [22]. There are two types of homomorphic encryption: partial homomorphic encryption and fully homomorphic encryption. In this work, partial homomorphic encryption, also known as Pailier methods [23], was used to improve efficiency with less computation and high security. In addition, partial homomorphic encryption is used for access control and authentication.

3. Related Work

Several studies were conducted in order to avoid malicious docker container image distribution as well as malicious forgery or alteration of images in the repository. The relevant studies are described below.

Q. Xu et al. propose a blockchain-based Decentralized Content Trust [24] with the possibility of extending studies on notary services. Once the docker image is verified, the signature is converted into a transaction, and the transaction is uploaded to bitcoin using the Carbon chain's unique library. The purpose was to ensure signed data denial blocking to provide decentralized services. However, while the service validates the signature of the docker image, no scans of the docker image are performed, making it difficult to determine whether the docker image is malicious.

J. Sun et al. [25] developed a Blockchain-based automatic container cloud security by performing a vulnerability check on the docker image and registering it on the Ethereum blockchain. The vulnerability checks on the docker image are unambiguous. The user's information and other data are also recorded, leading to flaws in the docker image and its signature. This determines the CVE linked with the vulnerability and provides precise information about docker images. In this study, the limitation is that personal information could be revealed when data were saved in the blockchain. Similarly, Y. Zheng et al. [26] introduced ZeroDVS, a secure container image based on inheritance graphs. This research aimed to find the vulnerabilities of the docker image through the layer of the docker image. The public container images were checked for inherited images, the relationship was examined, and the user was notified of any vulnerabilities. There is no mechanism for secure sharing in the proposed solution.

To secure applications, S.H. Han et al. [27] developed a container image access control architecture. Unauthorized users are prevented from accessing the container with basic user information. For ID authorization, each time the user ID is retrieved after downloading the container image, it is registered with the kernel policy administrator to grant ID-based privileges. The author claims that by using this strategy, unauthorized users will be unable to utilize the Docker image without authentication.

The Docker Image Vulnerability Diagnostic System (DIVDS) was developed by Soonhong Kwon et al. [11] to allow users to identify the vulnerability level of each docker image through the vulnerability evaluation process. The evaluation process is based on the combined relationship of vulnerable software packages and vulnerability information in a docker image. DIVDS assures a reliable docker-based application build environment by preventing users from downloading or uploading vulnerable docker images in a docker image repository. Similarly, Manish K A et al. [28] proposed architecture to ensure security for docker containers. The proposed architecture includes a plugin that uses a CI/CD pipeline to deploy the application and to ensure the security of the application. The architecture ensures security from the starting stage of application development to the deployment stage including plugin for docker build. The application is bundled in the form of images along with required libraries, pushing the images to a docker registry. The vulnerabilities in both static and dynamic resource allocations are validated. The entire implementation is automated without any manual interventions. The docker security levels' application mechanism was proposed by Vipin Jain et al. [29] to test the different platforms and measure the security attribute to access the resource.

Blockchain-based lightweight security framework named BlockchainBus was proposed by Joseph Doyle et al. [30] to secure virtual machines when migrating from one cloud to another. BlockchainBus is implemented using the HyperLedger solution and deployed on the Microsoft Azure cloud platform. The proposed framework was validated by comparing with the overall VM migration time and was determined as 2.36 s, whereas the overall VM migration time was 5 s. Similarly, a general call policy restriction method for the domestic operating system based on a docker container was proposed by Xu Xin et al. [31]. The proposed scheme provides practical security protection reinforcement for containers. An active defence mechanism was developed to filter and isolate container threats. The following security measures were considered in a study that Xu Xin et al. conducted: filtering system call, authorized installation, real-time signature verification, monitoring, resource control, access control and boundary protection.

Applying Zero Trust Containers Architecture (ZTCA) [32] to secure docker containers was proposed by Darragh Leahy et al. [33]. Initially, the authors investigated the security state of a default deployment of the docker container engine on Linux and analyzed how the Zero Trust containers Architecture can be extended beyond the networking to secure docker deployments. The author validated the ZTCA with the following five test scenarios: (a) grant user access to remote API, (b) prevent Container network attack, (c) review a docker image, (d) review an existing container and (e) deploy a privileged container. Similarly, Robail Yasrab [7] developed a mitigation strategy to secure two types of docker attacks, namely insider and outside attacks. The proposed solution was based on an access control methodology to ensure appropriate access management. In this method, the image maintainers ship the SELinux policy module and its images to the host platform. The policy models for an image are stated in the docker file and placed in the image metadata during creation of the image. The author claims that correct configuration and security policy ensure the greater security of a docker container.

Ferdinand Brasser et al. [34] developed a novel container security architecture named Trusted Container Extension (TCX). The proposed architecture ensured integrity and confidentiality for containers executed in the untrusted cloud. The TCX provided an extended integration to the docker container which was based on AMD Secure Encrypted

Virtualization (SEV) [35] and the kata container project [36]. TCX acts as a secure channel for communication so that the docker cannot distinguish between locally or remotely executed containers. The proposed architecture was validated with three benchmarks, namely (a) computational intensive workloads, (b) network intensive workloads and (c) memory intensive workloads. The author claims a performance impact of 5.77% and network throughput overhead of 22.1% for the NGINX webserver and overhead of 13.36% for the Apache webserver.

According to the literature, no framework specifies how to share the Docker image safely. This allows unauthorized users to edit the docker image and hence update the identity, allowing the attack to proceed through the docker image. This paper examined suitable user authentication for Docker images in order to safeguard their sharing, allowing only authorized users to view and share docker image data. To prevent successive docker images from being lost, the Secure Docker Image Sharing with Homomorphic Encryption and Blockchain (SeDIS-HEB) architecture was developed in this research which accounted for data stability, availability, and integrity.

4. Blockchain-Based Homomorphic Encryption (SeDIS-HEB Framework)

The proposed architecture uses Homomorphic encryption, authentication, and access control to provide docker images securely. The functionality implementation of a docker image with modifications made using the SeDIS-HEB system's existing docker architecture is shown in Figure 1. The SeDIS-HEB system consists of the following four components: the SeDIS-HEB Docker client, the Docker Engine and Docker hub, the IPFS peer-to-peer network, and the Ethereum blockchain peer-to-peer network. The existing Docker client module was modified to include homomorphic encryption authentication. SeDIS-HEB is made up of several submodules that work together to ensure the security of docker image upload, image share, and image download at the DI customer's request.

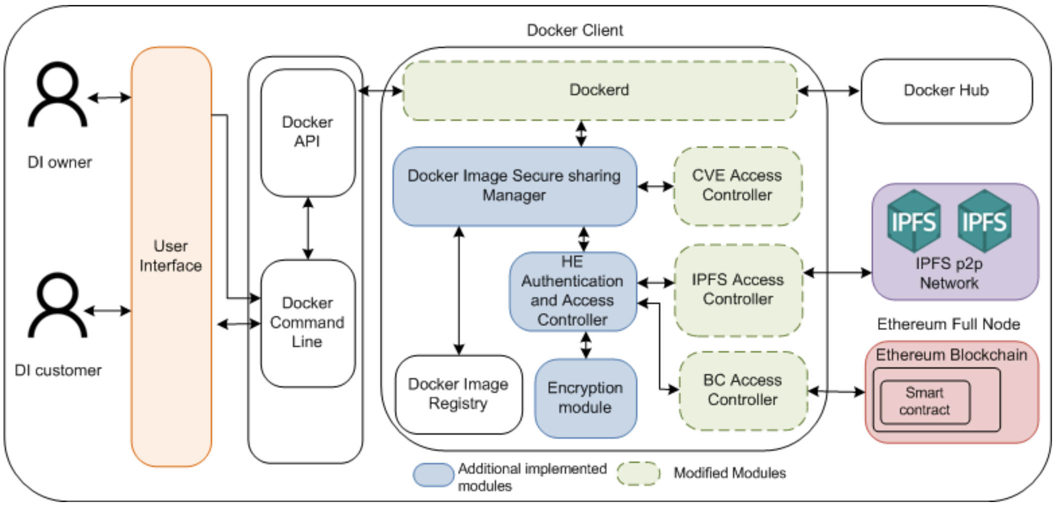


Figure 1. SeDIS-HEB System Architecture.

4.1. SeDIS-HEB Docker Client

The security components are integrated into the existing docker container design. The source code used is from the docker repository [37]. The modified source is updated in the following repository (https://github.com/seunggin/SDIS accessed on 25 July 2022). The following provides a detailed explanation of the three newly integrated modules:

Docker Image Secure Sharing Manager—The docker image secure manager ensures the following three logics: secure docker image upload for secure uploading of docker images, secure docker image share for sharing uploaded docker images, and secure docker images image download for download by the recipient. To make all the three logics possible, the docker manager ensures each controller, encryption module, and registry can be requested to perform the functions and pass on the resulting values.

HE Authentication & Access Controller—Homomorphic encryption, authentication, and access control are provided by the HE authentication and access controller. Upon receipt of essential information such as user data, docker image, CVE lists, HE authentication, and access controller module, the data are cross-checked and sent to the encryption module, where encryption is completed, and the image BCID is received. The image BCID controls access to the IPFS access controller.

Encryption Module—Paillier Homomorphic Encryption (PHE) [23], AES-256 symmetric encryption [38], and RSA-256 Asymmetric Encryption [21] were the three methods of encryption used in this module. The authentication and control access to the individual data were ensured using the Paillier homomorphic encryption technique. Before posting docker images and CVE lists to IPFS, AES-256 was used to encrypt them. To preserve the security of symmetric key data, RSA-256 asymmetric encryption was utilized to decrypt the docker image and CVE list hash values uploaded on top of the blockchain, as well as the encrypted metadata.

4.2. Docker Engine and Docker Hub

This paper used the existing Docker Engine and Docker Hub. Docker Engine is in charge of publishing docker images and managing containers, whereas Docker Hub [39] is in charge of publicly storing docker images.

4.3. IPFS P2P Network

The proposed architecture securely used IPFS to store the docker images and CVE lists. The user-requested files are sent to the user-registered node. The IPFS peer-to-peer (P2P) network is linked and connected with other peer nodes simultaneously to ensure the smooth transfer of the requested data.

4.4. Ethereum Blockchain P2P Network

Through the Smart Contract capability for data registration and invocation, the Ethereum blockchain P2P network grants access control to the docker image metadata.

4.5. Homomorphic Authentication via ImageBCID

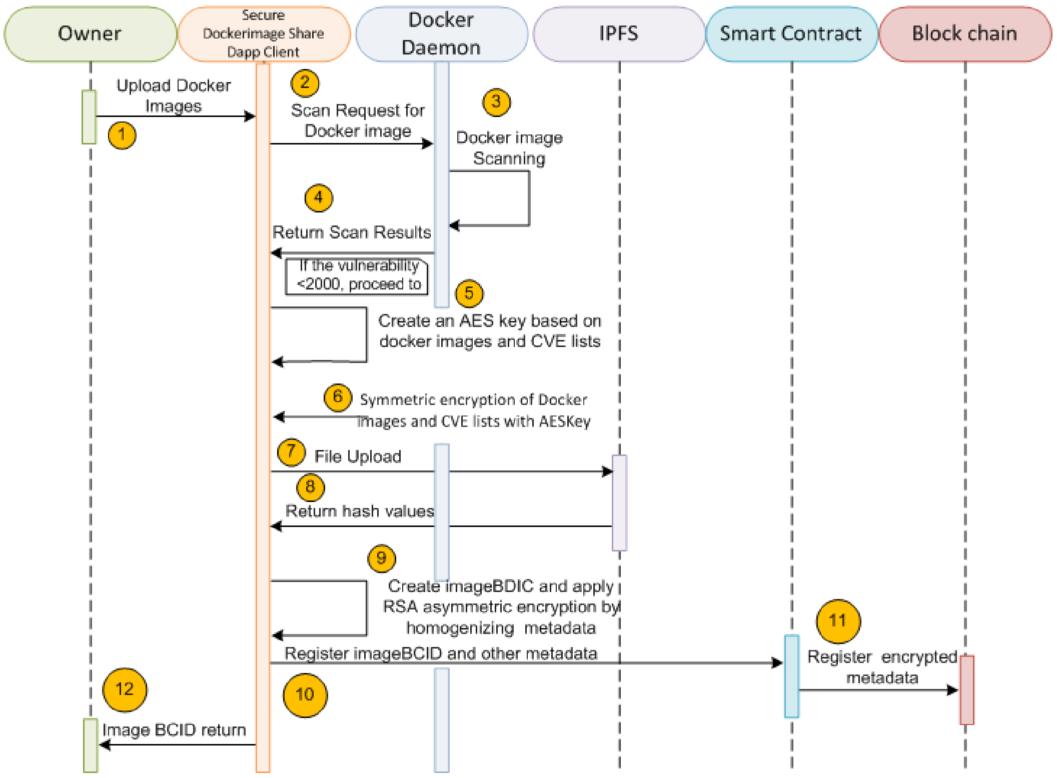
Fowler_Noll-Vo function process—The Fowler Noll Vo (FNV) hash function varies substantially with a small change in value. We can see in the above ImageBCID generating procedure that we first processed FNV hash functions before proceeding to homomorphic encryption. This solution requires the following four types of data: user information, hashes obtained after uploading the docker to IPFS, hashes obtained after uploading the CVE list to IPFS, and symmetric keys that encrypt the docker image and CVE list using the AES-256 algorithm. After string processing, the four values were transformed into integers using the FNV hash algorithm. Converting these values to integers allows them to be used in homomorphic encryption. The three docker image-related data cannot be modified during this operation; however, the user information provided by the user can only be submitted by those who know the information accurately. Even if this value is significantly changed, the FNV hash method yields a different output.

4.6. Homomorphic Encryption Authentication and Access Control with ImageBCID

Docker image sharing and downloading require authentication processes in order to gain the required docker image rights. The certification process produced two results. This occurs when authentication fails for the incorrect user but passes for the correct user. When a user fails to authenticate, all functions, regardless of process, are terminated, and the user is notified of the failure. It also restricts access to blockchain data, which is required to obtain IPFS docker images. We provided access control over actions via data for docker image calls on top of the blockchain for users that are correctly logged in. The DI Owner can share docker images with others during this process, and the DI Consumer can decrypt and use docker images that are encrypted and downloaded from the shared docker images.

Use Case1: Secure docker Image Upload—The first use case is to upload the docker images securely. The docker image owner utilizes this functionality to check for vulnerabilities before publishing the docker image. Then, the docker image and CVE list are uploaded after the docker image has been analyzed. The ImageBCID key value will then be returned. This is the most critical value for future Docker image downloads or sharing.

Each CVSS value is generated separately to create the CVE list. The corresponding values are then gathered, and access is limited based on CVSS ratings. As seen in Figure 2, the user initially requests that the Docker image be uploaded. When the command is received, the Docker Command Line performs the Docker Image Secure Sharing Manager's Secure Docker Image Upload logic (DISSM). The DISSM uses the Docker API to request a scan of the Docker image and then uses the CVE Access Controller to obtain the CVSS score of the CVE list. If the CVVS rating exceeds the threshold, the CVE access controller prevents the Docker image from being uploaded. Otherwise, just the necessary docker image and a list of CVEs are returned. The safe docker image upload pseudocode is depicted in code snippet 1.



Owner
Secure Dockerimage Share Dapp Client
Docker Daemon
IPFS
Smart Contract
Block chain
Upload Docker Images
1
2
Scan Request for Docker image
3
Docker image Scanning
4
Return Scan Results
If the vulnerability <2000, proceed to
5
Create an AES key based on docker images and CVE lists
6
Symmetric encryption of Docker images and CVE lists with AESKey
7
File Upload
8
Return hash values
9
Create imageBDIC and apply RSA asymmetric encryption by homogenizing metadata
Register imageBCID and other metadata
11
Register encrypted metadata
12
10
Image BCID return

Figure 2. Sequence flow of the secure docker image upload.

Once the vulnerability is identified using the CVE list, the docker image is symmetrically encrypted using AES-256 symmetric encryption and the CVE list. Using IPFS Access Controller, the encrypted picture is then uploaded to IPFS (IPFS-AC). The IPFS-AC generates the corresponding hash value and returns it to the SeID client. As previously shown, the user can submit a Docker image and create an ImageBCID containing all the data. Because user data are hashed with FNV and consists of an ID and password, there is no risk of the value leaking even if it is later recovered. Prior to Homomorphic encryption, the FNV function was used to generate hashes and Homomorphic encryption. This approach creates value using the Paillier Homomorphic encryption technique, which allows encryption with the public key. The imageBCID is formed by combining the values produced via homomorphic encryption. The value is then used as the blockchain's key value in order to obtain further data and authenticate users. The ImageBCID is then registered on the blockchain with key values using BC Access Controller smart contracts. After that, the ImageBCID is returned to the user.

```
if docker score = assigned score then
deliver the corresponding doker image and CVE list
else
return Null
end if
return CVVS score
end function
CVEList  = CVE_AccessControl (Cvelist.json)
AESKey = sha256 (marshaIJSON (ownerInfo + CVEList)) //
    Encrypting docker image and CVE Lists
enDimage, enCVE = AES256Encryption (AESKey, Docker_Image,
    CVElist) //file upload to IPFS
homoCVE, homodimage, homoAESKey, homoOwnerInfo =
    homoRSAEncryption (homoPublicKey, FNVHash (CVE_IPFSHash),
          FNVHash (Dimage_IPFSHash), FNVHash(AESKey), FNVHash(
    ownerInfo)) //return the hash value
ImageBCID = makeBCID_by_homoAdd (homoCVE + homoDimage +
    homoAESKey + homoOwnerInfo) // Image BCID creation and
    blockchain registration
rsaCVE_IPFSHash, rsaDimage_IPFSHash, rsaAESKey = RSAEncryption
              (ownerPublickey, CVE_IPFSHash, Dimage_IPFSHash,
    AESKey)
registertoBlockchain (ImageBCID, rsaCVE_IPFSHash,
    rsaDimage_IPFSHash, rsaAESKey)
```

Code Snippet 1: Secure docker image upload (Input: Address, Docker Image name, CVEs. Output: upload secure docker image, check for vulnerabilities, CVE list, ImageBCID key value.

4.7. Use Case2: Secure Docker Image Share

Secure Docker Image Share presupposes that a docker image is uploaded as indicated in Figure 3, in which the owner of the docker image has a corresponding ImageBCID, and the user who wishes to share the docker image has homomorphically encrypted it. This technique enables docker image owners to share the docker images with requested users safely. The new ImageBCID is generated, and the same is shared with the user. As illustrated in Figure 3, the metadata are retrieved from the docker image through blockchain via the BC access controller and image BCID. The retrieved metadata will be used for authentication using FNV hash functions and homomorphic encryption. Once the data are encrypted, the image is authenticated using the personal password of the docker image owner. Further decryption is carried out if the two values are matched. The safe docker image share pseudo code is depicted in code snippet 2.

To share these values again, it undergoes asymmetric encryption using the DI Consumer's public key. New values are prepared to be registered in the blockchain as a result of this. The data are then registered with the new New Image BCID as the key value by the BC Access Controller. It then returns the user the New Image BCID.

4.8. Use Case3: Secure Docker Image Download

Secure Docker Image Download works on the assumption that a docker image has been uploaded and that the image BCID is known. Figure 4 depicts the full process of the DI Owner downloading the image from the DI Consumer after it has been shared. To begin, the ImageBCID is transmitted to request the download of the corresponding docker image. The SeDIS-HEB Docker client imports Docker image metadata that matches the ImageBCID. Once the matching process is carried out, the docker metadata is decrypted by the DI consumer with the private key transmitted through the FNC. Like the previous homomorphic encryption authentication, the downloader combines the rest of the metadata with their password and proceeds with the authentication procedure. Users who complete the authentication process will be sent encrypted docker images and CVE lists via decrypted data. The docker image and CVE list are then re-symmetrized using asymmetric decrypted symmetric keys. It then returns the user the docker image and CVE list. The pseudocode is illustrated in code snippet 3.

```
CVE_IPFSHash, Dimage_IPFSHash, AESKey = RSADecryption (
    ownerPrivateKey, rsaCVE_IPFSHash, rsaDimage_IPFSHash,
    rsaAESKey) // CVE hash, image hash, AES key
homoCVE, homoDimage, homoAESKey, homoOwnerInfo =
    homoRSAEncryption    (homoOwnerPublicKey, FNVHash (
    CVE_IPFSHash), FNVHash (Dimage_IPFSHash), FNVHash (AESKey),
     FNVHash (ownerInfo)) // Verification with access control
function Verification Result (FHE Access control) Verification
     Result = FHE_AccessControl (imageBCID, homoCVE, homoDimage
    , homoAESKey, homoOwnerInfo) // Verification with access
    control
function Verification Result (FHE Access control) Verification
     Result = FHE_AccessControl (imageBCID, homoCVE, homoDimage
    , homoAESKey, homoOwnerInfo)
If verification Result == false then
return Null
else
newImageBCID = makeImageBCID (homoConsumerInfo, homoCVE,
    homoDimage, homoAESKey)
end if
return verification Result
end function
rsaConCVE_IPFSHash, rsaConDimage_IPFSHash, rsaConAESKey =
    RSAEncryption (consumerPublicKey, CVE_IPFSHash,
    Dimage_IPFSHash, AESKey) // register the newly generated
    BCID with blockchain
registertoBlockchain (newImageBCID, rsaConCVE_IPFSHash,
    rsaConDimage_IPFSHash , rsaConAESKey)
```

Code Snippet 2: Secure docker image share (Input: Request docker metadata. Output: New image BCID).

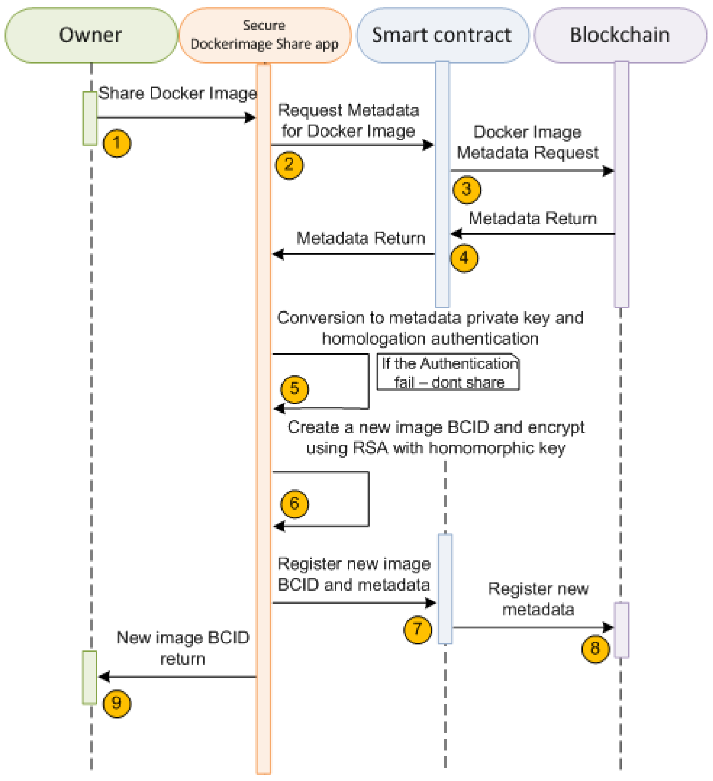

Owner
Secure Dockerimage Share app
Smart contract
Blockchain
Share Docker Image
1
Request Metadata for Docker Image
2
Docker Image Metadata Request
3
Metadata Return
4
Metadata Return
Conversion to metadata private key and homologation authentication
If the Authentication fail – dont share
5
Create a new image BCID and encrypt using RSA with homomorphic key
6
Register new image BCID and metadata
7
Register new metadata
8
New image BCID return
9

Figure 3. Sequence flow of the secure docker image share.

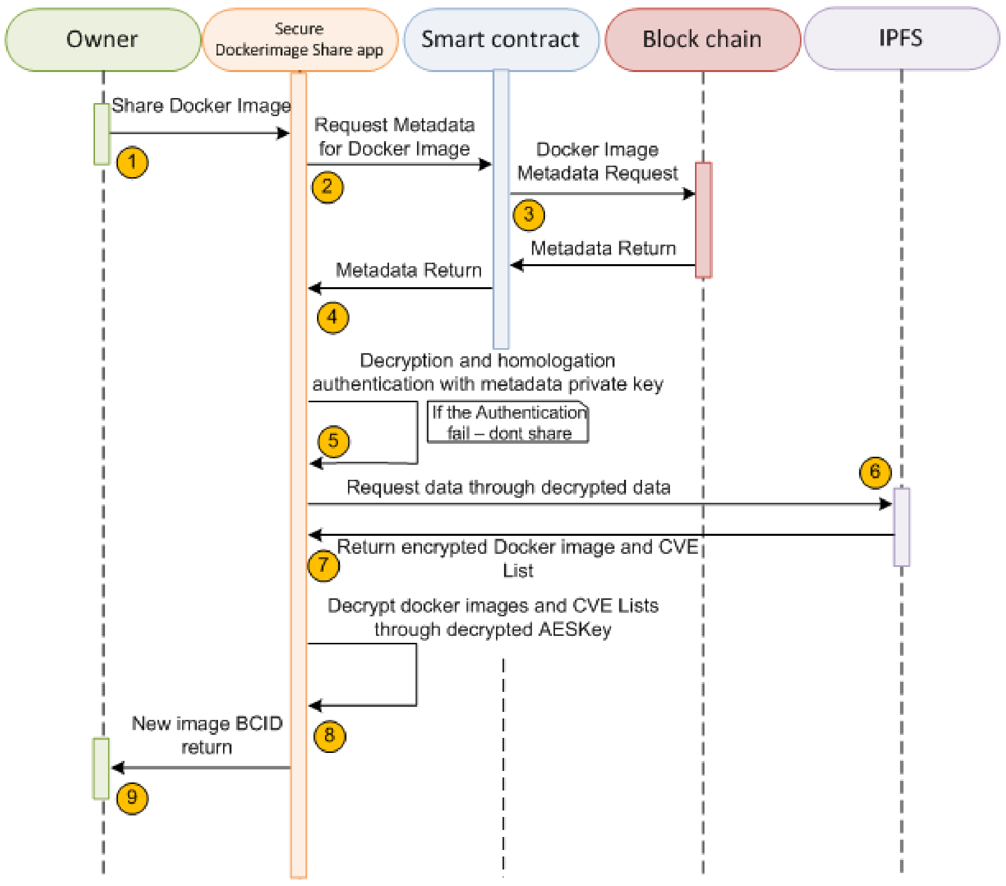

Owner
Secure Dockerimage Share app
Smart contract
Block chain
IPFS
Share Docker Image
1
Request Metadata for Docker Image
2
Docker Image Metadata Request
3
Metadata Return
Metadata Return
4
Decryption and homologation authentication with metadata private key
If the Authentication fail – dont share
5
6
Request data through decrypted data
Return encrypted Docker image and CVE List
7
Decrypt docker images and CVE Lists through decrypted AESKey
8
New image BCID return
9

Figure 4. Sequence flow of the secure dockerimage download.

```
request Docker Image Metadata (newImageBCID)
homoCVE, homoDimage, homoAESKey, homoConsumerInfo =
    homoRSAEncryption (homoConPublicKey, FNVHash (CVE_IPFSHash)
    ,    FNVHash (Dimage_IPFSHash), FNVHash (AESKey), FNVHash(
    ownerInfo)) //homomorphic encryption
VerificationResult = FHE_AccessControl (newImageBCID, homoCVE,
     homoDimage, homoAESKey, homoConsumerInfo) //Verification
    of results
function Verification Result (FHE Access control)
Verification Result = FHE_AccessControl (newImageBCID, homoCVE
    ,
homoDimage, homoAESKey, homoConsumerInfo)
If verification Result == false then
return Null
else
enCVE, enImage = downloadFromIPFS (CVE_IPFSHash,
    Dimage_IPFSHash)
end if
return verification Result
end function
CVElist, Dimage = AES256Decrypt (AESKey, enCVE, enDimage) //
    decryption using AES
```

Code Snippet 3: Secure docker image download (Input: Request docker metadata. Output: Docker image, CVE List).

5. Verification and Validation

Verification and Validation are carried out using the docker that does not provide security guarantees for vulnerable software packages installed within the docker image. This paper applied the proposed SDIS HEB to the docker environment when uploading, sharing, or downloading docker images. To demonstrate the applicability of the SDIS HEB, we verified and validated the SDIS HEB works correctly. SDIS HEB security was ensured with the following quality properties.

5.1. Quality Properties

The quality properties are studied in this section to verify and validate the proposed security mechanism to ensure the docker image is securely shared among the user.

Homomorphic-based authentication—The docker image is not shared with anyone who is not the Data Owner of that docker image or who is not allowed to share it. By collecting the docker image provided by other attackers, this authentication stops the attacker from polluting other people's docker containers. This research provides authentication using ImageBCID, created via homomorphic encryption. Even if an unauthorized user attempts to download with ImageBCID, the download will fail since ImageBCID is not produced using the personal information they provided during the authentication process.

Docker Image Content-Based Access Control with Homomorphic Encryption—In this research, we used ImageBCID to control metadata access for Docker images stored on the blockchain. The docker image stored on IPFS cannot be accessible if no metadata is imported. The role of ImageBCID is to offer access control to blockchain data activities. ImageBCID contains metadata from Docker images as well as personal passwords that are encrypted after the hash, granting them access to data in the blockchain via that value. This document stops users from computing their data if they fail to authenticate in an Encryption module. In other words, the blockchain's access control fails to retrieve docker images and CVE lists from IPFS.

Confidentiality of data uploaded to blockchain and IPFS—The system then uses symmetric and asymmetric encryption to encrypt data written on the blockchain. Symmetric encryption was employed during the encryption operation for CVE lists and docker images with large data volumes. Metadata is intended to discourage users from using IPFS, even if they download it without authorization. Symmetric keys for decrypting metadata and hash values for retrieving data from IPFS are also encrypted and stored on the blockchain using asymmetric keys. As a result, even if the data are obtained by someone else, they cannot be used. During the ImageBCID creation process, the user sends a homomorphically encrypted value to the data owner instead of the original password of the Data consumer and then constructs a new ImageBCID using the encrypted value.

Denial containment of Docker image metadata—If the docker image metadata are corrupted, obtaining the related docker image is generally difficult. On the other hand, this approach prevents metadata deterioration since it maintains metadata about Docker images in the blockchain. Because of its structure, blockchain involves modifying the books of most nodes to corrupt data, which is essentially impossible. If an image posted to this system is infected, it is hard to dispute that the docker image was uploaded.

Stability for Docker Image—Other prior studies identified a lack of research on docker image storage. Therefore, this effort provided availability for this component through distributed storage. Because encrypted docker images and CVE lists are distributed over IPFS, local repositories utilizing existing containers may overcome the disadvantages of disappearing and data disappearing simultaneously, ensuring stability.

Integrity to Docker Image—Because IPFS returns hash values based on the file's contents, the hash values of the file change as the file's content changes. After the encryption procedure in this system, the hash values are placed on the blockchain. Following that, the encrypted hash values are re-imported, assuring the integrity of the docker image and the CVE list.

5.2. Comparative Analysis of the Proposed Work

As shown in Table 1, five related studies and the systems provided in this paper were divided and compared based on six items. We show that there were deficiencies in each related study and that the system structure proposed in this paper provides each of the quality attributes discussed in Section 3. Each property lacking in existing relevant studies is guaranteed through the proposed architecture. We provided non-repudiation for docker image metadata via blockchain, docker image vulnerability check with docker scan, user authentication for docker images, confidentiality guarantees for each docker image and metadata, access control for docker images, and stability in storing docker images reliably.

Table 1. Comparative analysis of SDIS HEB with landmark methods.

Core Security Essentials	Notary	Decentralized Content Trust	Security System	Zero DVS	Container Image Access Control	SDIS HEB
Blockade and Docker Image Information	NO	YES (bitcoin)	YES (Ethereum)	NO	NO	YES (Ethereum)
Vulnerability checking for docker image	NO	NO	YES	YES (image check inheritance)	NO	YES (Docker Scan)
Authentication procedures for docker image unique	NO	NO	NO	NO	NO	YES (ImageBCID)
Ensure confidentiality of docker information	NO	NO	NO	NO	NO	YES (AES, RSA)
Docker image data access control	NO	NO	NO	NO	YES (Container Access control)	YES (ImageBCID)
Stability for docker image	NO	NO	NO	NO	NO	YES (IPFS)

5.3. Evaluation and System Performance Analysis

We can observe that only metadata computations have been performed, with no actual file uploads or downloads. Figures 5–7 depicts the time spent in the sharing procedure.

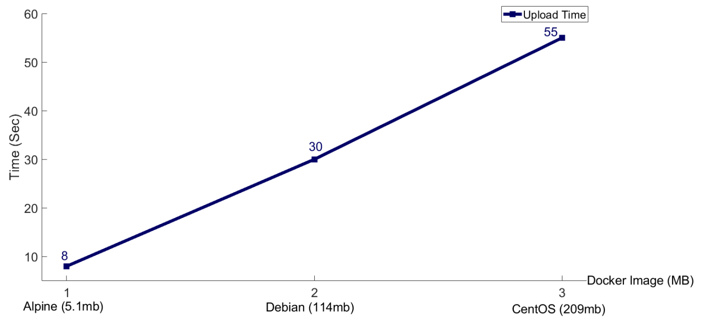


Figure 5. Comparison of upload time with Alpine, Debian and CentOS.

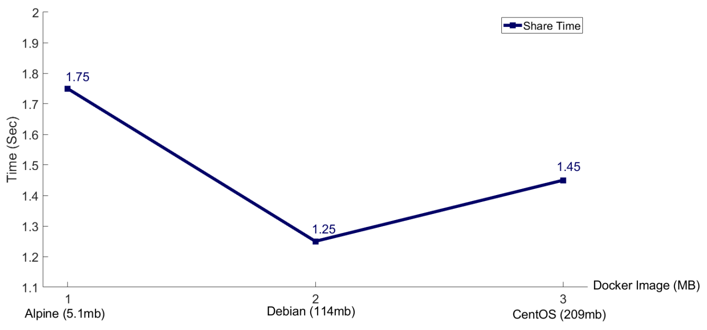


Figure 6. Comparison of shared time with Alpine, Debian and CentOS.

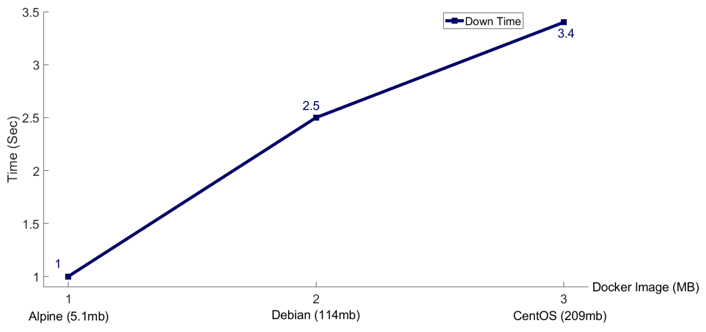


Figure 7. Comparison of download time with Alpine, Debian and CentOS.

The results show that alpine, Debian, and centos were uploaded from the official repository. However, Ubuntu had multiple flaws in the image, causing the upload to fail. Furthermore, ImageBCID was issued for all images using homomorphic encryption in a relatively short period of time using the paillier method. Likewise, wrongly supplied user information can be noticed to keep the user away from the data.

While reusing the docker image, the problem we face is that when the user downloads the data randomly and while sharing with the other user, our proposed work does not block the data flow. This leads to a lack of prevention of the unauthorized reuse of docker images. Currently, we are developing a Digital Right Management (DRM)-based prevention mechanism to prevent the unauthorized reuse of docker images.

6. Conclusions

This study proposed a distributed framework named Safe Docker Image Sharing with Homomorphic Encryption and Blockchain (SeDIS-HEB) for container image security to solve the security issues in one of the most extensively used infrastructures in cloud services. The following major findings were obtained:

(a) In the proposed frameworks, the information about the Docker image owner and docker image consumer users is accessed only by authorized users. This is achieved by providing a control scheme approach through authentication procedures and blockchain data through homomorphic encryption.
(b) While storing the docker image in IPFS, the confidentiality and integrity of data are ensured through asymmetric encryption.
(c) Homomorphic encryption allows users to secure their privacy by using features that can be computed by encrypting DI Consumer's privacy during docker image sharing.

The SeDIS-HEB architecture combines cutting-edge decentralized technologies such as IFPS and blockchain as core kernels. A secure scan will check for docker images on image upload to discover vulnerabilities in docker images, ensuring docker image metadata denial-of-service protection. We also provided authentication and access control using ImageBCID, which was constructed using homomorphic encryption, and privacy for the remaining data using various asymmetric and symmetric encryption methods. Furthermore, we are introducing the unauthorized reuse of docker images utilizing Digital Right Management (DRM) technology to improve the security of the docker image.

Author Contributions: Conceptualization, V.K.K., S.Y. and D.M.; methodology, D.M., S.Y.; software, S.Y.; validation, V.K.K., S.J., D.M. and R.S.; formal analysis, V.K.K.; investigation, D.M.; resources, E.C.; data curation, E.C.; writing—original draft preparation, V.K.K. and S.Y.; writing—review and editing, V.K.K. and S.Y.; visualization, D.M. and E.C.; supervision, D.M.; project administration, D.M.; funding acquisition, D.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. 2020R1A6A 1A03046811). This work was supported by the National Foundation of Korea (NRF) grant funded by the Korea government (Ministry of Science and ICT (MIST)) (No. 2021R1A2C209494311).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bernstein, D. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Comput.* **2014**, *1*, 81–84. [CrossRef]
2. Burniske, C. Containers: The Next Generation of Virtualization? 2016. Available online: https://ark-invest.com/articles/analyst-research/containers-virtualization/ (accessed on 25 July 2022).
3. Rodriguez, M.A.; Buyya, R. Container-based cluster orchestration systems: A taxonomy and future directions. *Softw. Pract. Exp.* **2019**, *49*, 698–719. [CrossRef]
4. Merkel, D. Docker: lightweight linux containers for consistent development and deployment. *Linux J.* **2014**, *239*, 2.
5. Boettiger, C. An introduction to Docker for reproducible research. *ACM SIGOPS Oper. Syst. Rev.* **2015**, *49*, 71–79. [CrossRef]
6. Tunde-Onadele, O.; He, J.; Dai, T.; Gu, X. A study on container vulnerability exploit detection. In Proceedings of the 2019 IEEE International Conference on Cloud Engineering (IC2E), Prague, Czech Republic, 24–27 June 2019; pp. 121–127.
7. Yasrab, R. Mitigating docker security issues. *arXiv* **2018**, arXiv:1804.05039.
8. Rad, B.B.; Bhatti, H.J.; Ahmadi, M. An introduction to docker and analysis of its performance. *Int. J. Comput. Sci. Netw. Secur. (IJCSNS)* **2017**, *17*, 228.
9. Rajalakshmi, A.; Lakshmy, K.; Sindhu, M.; Amritha, P. A blockchain and ipfs based framework for secure research record keeping. *Int. J. Pure Appl. Math.* **2018**, *119*, 1437–1442.
10. MITRE. CVE Records. Available online: https://www.cve.org/ResourcesSupport/Resources (accessed on 17 June 2022).
11. Kwon, S.; Lee, J.H. Divds: Docker image vulnerability diagnostic system. *IEEE Access* **2020**, *8*, 42666–42673. [CrossRef]
12. Anchore. *Docker Image Security*; Anchore: Santa Barbara, CA, USA, 2022.
13. Clair. Clair—Static Analysis of Vulnerabilities. 2020. Available online: https://github.com/quay/clair (accessed on 23 June 2022).

14. Snyk. Snyk Engine. 2019. Available online: https://snyk.io/product/open-source-security-management/ (accessed on 12 March 2019).
15. Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Bus. Rev.* **2008**, 21260. Available online: https://www.researchgate.net/publication/228640975_Bitcoin_A_Peer-to-Peer_Electronic_Cash_System (accessed on 12 March 2019).
16. Naz, M.; Al-zahrani, F.A.; Khalid, R.; Javaid, N.; Qamar, A.M.; Afzal, M.K.; Shafiq, M. A secure data sharing platform using blockchain and interplanetary file system. *Sustainability* **2019**, *11*, 7054. [CrossRef]
17. Mohanta, B.K.; Panda, S.S.; Jena, D. An overview of smart contract and use cases in blockchain technology. In Proceedings of the 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Bengaluru, India, 10–12 July 2018; pp. 1–4.
18. Vujičić, D.; Jagodić, D.; Ranđić, S. Blockchain technology, bitcoin, and Ethereum: A brief overview. In Proceedings of the 2018 17th International Symposium Infoteh-jahorina (Infoteh), East Sarajevo, Bosnia and Herzegovina, 21–23 March 2018; pp. 1–6.
19. Buterin, V. *A Next-Generation Smart Contract and Decentralized Application Platform*; White Paper; nft2x.com: New York, NY, USA, 2014; Volume 3, pp. 1–2.
20. Solidity. Object-Oriented, High-Level Language. 2016. Available online: https://docs.soliditylang.org/en/v0.8.11/ (accessed on 2 May 2022).
21. Simmons, G.J. Symmetric and asymmetric encryption. *ACM Comput. Surv. (CSUR)* **1979**, *11*, 305–330. [CrossRef]
22. Ogburn, M.; Turner, C.; Dahal, P. Homomorphic encryption. *Procedia Comput. Sci.* **2013**, *20*, 502–509. [CrossRef]
23. Koç, Ç.K.; Özdemir, F.; Özger, Z.Ö. Paillier Algorithm. In *Partially Homomorphic Encryption*; Springer: Cham, Switzerland, 2021; Volume 20, pp. 95–105.
24. Xu, Q.; Jin, C.; Rasid, M.F.B.M.; Veeravalli, B.; Aung, K.M.M. Blockchain-based decentralized content trust for docker images. *Multimed. Tools Appl.* **2018**, *77*, 18223–18248. [CrossRef]
25. Sun, J.; Wu, C.; Ye, J. Blockchain-based Automated Container Cloud Security Enhancement System. In Proceedings of the 2020 IEEE International Conference on Smart Cloud, Washington, DC, USA, 6–8 November 2020; pp. 1–6.
26. Zheng, Y.; Dong, W.; Zhao, J. ZeroDVS: Trace-ability and security detection of container image based on inheritance graph. In Proceedings of the IEEE 5th International Conference on Cryptography, Security and Privacy, CSP 2021, Zhuhai, China, 8–10 January 2021; pp. 186–192.
27. Han, S.H.; Lee, H.K.; Lee, S.T.; Kim, S.J.; Jang, W.J. Container Image Access Control Architecture to Protect Applications. *IEEE Access* **2020**, *8*, 162012–162021. [CrossRef]
28. Abhishek, M.K.; Rao, D.R. Framework to Secure Docker Containers. In Proceedings of the 2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4), London, UK, 29–30 July 2021; pp. 152–156.
29. Jain, V.; Singh, B.; Choudhary, N. Audit and Analysis of Docker Tools for Vulnerability Detection and Tasks Execution in Secure Environment. In Proceedings of the International Conference on Emerging Technologies in Computer Engineering, Jaipur, India, 4–5 February 2022; pp. 654–665.
30. Doyle, J.; Golec, M.; Gill, S.S. Blockchainbus: A lightweight framework for secure virtual machine migration in cloud federations using blockchain. *Secur. Priv.* **2022**, *5*, e197. [CrossRef]
31. Xu, X.; Zhang, Y.; Hao, Y.; Jiang, Y.; Geng, M. Research of Container Security Reinforcement Multi-Service APP Deployment for New Power System on Substation. In Proceedings of the 2022 4th Asia Energy and Electrical Engineering Symposium (AEEES), Chengdu, China, 25–28 March 2022; pp. 945–949.
32. Kindervag, J.; Balaouras, S. No more chewy centers: Introducing the zero trust model of information security. *Forrester Res.* **2016**, *3*, 1–15.
33. Leahy, D.; Thorpe, C. Zero Trust Container Architecture (ZTCA): A Framework for Applying Zero Trust Principals to Docker Containers. In Proceedings of the International Conference on Cyber Warfare and Security, Albany, NY, USA, 17–18 March 2022; Volume 17, pp. 111–120.
34. Brasser, F.; Jauernig, P.; Pustelnik, F.; Sadeghi, A.R.; Stapf, E. Trusted Container Extensions for Container-based Confidential Computing. *arXiv* **2022**, arXiv:2205.05747.
35. Kaplan, D. *Protecting VM Register State with SEV-ES*; White Paper; 2017. Available online: www.amd.com (accessed on 25 July 2022).
36. Kata. Kata Containers. 2017. Available online: https://katacontainers.io/ (accessed on 25 July 2022).
37. Docker. Docker. 2013. Available online: https://github.com/docker/docker.github.io (accessed on 25 July 2022).
38. Abdullah, A.M. Advanced encryption standard (AES) algorithm to encrypt and decrypt data. *Cryptogr. Netw. Secur.* **2017**, *16*, 1–11.
39. Yadav, S.P.; Agrawal, K.K.; Bhati, B.S.; Al-Turjman, F.; Mostarda, L. Blockchain-based cryptocurrency regulation: An overview. *Comput. Econ.* **2020**, *59*, 1659–1675. [CrossRef]