electronics

Article

Blockchain-Based Employment Contract System Architecture Allowing Encrypted Keyword Searches

Sanghui Yeom, Seungyeon Choi, Jeonghee Chi and Soyoung Park *

Department of Computer Science and Engineering, Konkuk University, Seoul 05029, Korea; ysh2911@konkuk.ac.kr (S.Y.); bzt2971@konkuk.ac.kr (S.C.); jhchi@konkuk.ac.kr (J.C.)
* Correspondence: soyoungpark@konkuk.ac.kr; Tel.: +82-2-450-0482

Abstract: We introduce a blockchain-based online employment contract system to protect the rights and interests of both employees and employers. In the proposed model, an employer and a worker can interactively create a new electronic online contract, and the mutually signed contract is saved on a contract blockchain so that the contract becomes certifiable but cannot be forged by the contract signers. In particular, the blockchain in our system provides transactional privacy to protect sensitive personal information such as social identifier, contact information, income, and so forth, contained in the contract. Since a remote cloud server must provide not only secure storage, accessibility, and availability of all signed contracts, but also increased security in the server, we propose a new encrypted keyword search mechanism with enhanced search accuracy. Each contract is associated with encrypted keywords generated from the names of contractual parties and must remain confidential and anonymous even to the server. Although, the contracts must always be accessible by the contract signers, only the cloud server should be able to retrieve each user's contract without decrypting the contract or identifying the contract signer. To meet these requirements, we propose a new encrypted keyword search mechanism based on Gentry's homomorphic encryption technology; the server can find each user's contract when two encrypted arbitrary keywords are homomorphic to each other. Since the keywords in the proposed system are based on person names or business names, they are easily predictable, and, thereby, many synonyms for a keyword can exist. Therefore, the proposed encrypted keyword search takes into account not only the keywords but also the ownership of each contract; in this way, the proposed search scheme is secure against a keyword guessing attack and provides strong search accuracy against the keyword synonyms. As a result, users can only access their own contracts, and the cloud server can exactly retrieve the requester's contracts. Implementations for the proposed system and corresponding analysis on its security and simulated performance are provided.

Keywords: employment contract; blockchain; homomorphic encryption; keyword; encrypted search

Citation: Yeom, S.; Choi, S.; Chi, J.; Park, S. Blockchain-Based Employment Contract System Architecture Allowing Encrypted Keyword Searches. *Electronics* **2021**, *10*, 1086. https://doi.org/10.3390/electronics10091086

Academic Editors: Diana Berbecaru, Detlef Hühnlein and Costin Badica

Received: 7 April 2021
Accepted: 29 April 2021
Published: 4 May 2021

1. Introduction

An employment contract or a labor contract (hereinafter referred to as a contract) that specifies the working conditions and requirements is essential to protect the rights and interests of employers and employees. Recently, the types of work have been diversified; international work is increasing due to globalization, and part-time work and temporary work are also increasing due to slow economic growth and high unemployment rate. Contracts for part-time workers and temporary workers are very important, even though they are temporary and renewed frequently. However, unlike the contracts for regular workers, those for part-time and temporary workers have been improperly executed. Even when contracts are executed, employers often use non-standard forms that may not be sufficient for legally binding. In the absence of well-defined work requirements and conditions, unfair labor practices such as nonpayment of wages can occur. For example, in one survey on improper wages for part-time jobs, wages had been overdue for 59.9% of

1546 participants, and unpaid for 27.2% [1]. Therefore, a practical employment contract system that can handle various types of employment and working conditions is necessary.

In general, contracts can be classified into three types: written contracts, digital contracts, and electronic contracts. Written contracts are most commonly used off-line, but they may not be secure enough for keeping personal information, preventing from forgery, and loss. PDF (Portable Document Format) contracts, which are widely used online in recent years, are the most representative examples of digital contracts. Digital contracts have the advantages of being portable and convenient, maintaining the exact appearance of document, and working online. However, the PDF document is still an image, so it is susceptible to contract forgery; even if it is signed, the integrity of the contract content is not guaranteed because the signature is also an image independent of the contract content.

Unlike digital contracts, electronic contracts involve cryptographic technologies including encryption, digital signature, and digital right management that ensure strong security. These technologies support contract confidentiality, non-forgery, and contract verifiability as well as contractor authentication.

In this paper, we introduce a blockchain-based electronic contract system. Contracts contain personally sensitive information such as identifiers, contact information, and wages. Therefore, we propose here a secure and reliable online contract system that satisfies the following security requirements: (1) confidentiality, (2) non-forgery, (3) verifiability, (4) secure accessibility and availability, and (5) contractor anonymity.

Our proposed model will enable a pair of an employer and a worker to create a new contract online. A contract blockchain manages contracts that have been signed by both sides in hashes the system records to the blockchain. The general advantage of using blockchain is that any information recorded on the blockchain can be publicly verified by anyone on the blockchain network, but labor contracts should not be publicly broadcast over networks. Therefore, in our system, only the hash of the contract is communicated over the blockchain network, and all of contractual parties perform a proof of work for a block containing their contracts to update the blockchain. The contract signers can validate the transaction hashes of their own contracts during the blockchain generation and always verify their contracts using the blockchain. Therefore, the proposed blockchain model serves as the ledger of contract creation record while effectively preserving the personal information of contracts.

For secure and efficient management of the signed contracts, each contract signer generates an encrypted contract and two or three encrypted keywords using the names of the contractual parties. The encrypted contracts are stored with the encrypted keywords on a cloud server that users can access using the keywords; only the contract owner can decrypt his or her contract. This process ensures both contract confidentiality and contractor anonymity in the cloud server; the server can never know who made a contract with whom. To ensure that only the cloud server can retrieve each user's contract, and only with encrypted keywords given by the user, we propose a new encrypted keyword search mechanism that considers both the contract keyword and the contract ownership. Using Gentry's homomorphic encryption technology, the server can determine if any two encrypted keywords are homomorphic to each other: a user generates and sends a newly created encrypted keyword to request his or her contract on the server, and then, the server compares the given keyword to all keywords in the list. The server can find all contracts that are matched with the same keyword as the given keyword. By the way, the keywords in the proposed system are generated from person names or business names, and thus, the keywords can be easily predicted and many keyword synonyms can exist. The server has to sift out only the requester's contract among the many contracts with the same keyword, and thus, the proposed search mechanism also uses a contract access token to verify if the requester is the contract owner. As a result, the proposed search model exactly retrieves the requester's contracts; at the same time, it is secure against a keyword guessing attack and accurate against the keywords synonyms.

The main achievements of our present work are as follows:

- a secure, reliable, and practical online employment contract system architecture using a blockchain that preserves transactional privacy is introduced.
- the contract management architecture using a cloud server which is secure and preserves personal information is proposed.
- enhanced encrypted keyword searching which is secure against keyword guessing attacks and accurate against keyword synonyms is proposed.

The rest of this paper proceeds as follows. In Section 2, we review related works. In Section 3, we briefly describe the cryptographic prerequisite closely associated with our system and provide concrete protocols and algorithms to implement our system. We analyze the security and efficiency of the proposed system in Section 4. Finally, we discuss and conclude our paper in Section 5.

2. Related Works

2.1. Blockchain-Based Smart Contracts

Smart contracts, introduced by Nick Szabo [2], are computer transaction protocols that allow contracts to be entirely written and signed online. They satisfy all requirements of general written contracts but minimize certain complications of the process such as abnormal exceptions and unreliable intermediaries. However, the initial model was not secure against contract forgery, contract integrity, and contract confidentiality.

Satoshi Nakamoto introduced blockchain technology in 2008 [3]. The blockchain is a distributed and shared database that maintains a continuously increasing list of ordered blocks containing users' transactions; it is built based on various cryptographic technologies including hash, digital signature, and consensus protocols such as proof of work and a proof of stake. The blockchain is updated by its previous block hash, the current block's Merkle tree root hash, and a random nonce satisfying a predefined system difficulty, and all data recorded in the blockchain can be publicly validated just by hash comparison. Therefore, it guarantees data integrity and unforgeability at the same time; it also provides undeniability of the transaction generation because every transaction is digitally signed.

The blockchain solves the security issues of online smart contracts, and various blockchain-based smart contract platforms have been developed. Ethereum by V. Buterin [4] is a global and distributed platform that is operated by a public blockchain, so all users of the blockchain network share all data. Transactions are not confidential to users, although users are anonymous. Ethereum is typically used for token generation and stock issuance, etc.

Hyperledger Fabric [5] was also developed as an enterprise blockchain. Unlike Bitcoin and Ethereum, it is a private blockchain that only authorized users can participate in. Because only some nodes perform any given smart contract, multiple contracts can be executed in parallel. Authorized users can be identified, but transactions are confidential to unauthorized users. Hyperledger Fabric is widely used for various applications such as personal health records [6], cooperation between education and industry [7], and a government funding tracking system [8]. Blockchain technology has been also widely applied for Industry 4.0, such as secure management of healthcare information [9], efficient and transparent management of industrial Internet of Things data [10], and fair recruitment and human resource management [11].

Related to employment contracts, Pinna and Ibba [12] suggested a blockchain-based temporary employment contracts system. The proposed decentralized employment system uses blockchain to securely and transparently manage the entire process of job opening of employers, job applying of workers, hiring, and payment. Lallai et al. [13] provided the software development methodology to build a decentralized application for the blockchain-based temporary employments. The systems mentioned above focus on the transparent management of the employment progress and process, but our system focuses on the secure management of the contract writing, creation, storage, and retrieval.

Because all transactions are distributed and shared publicly or privately over the blockchain network, they are exposed to any or all authorized users on the blockchain; thus, transactional privacy is not still preserved. Kosba et al. [14] proposed a privacy-

preserving smart contract system that does not store financial transactions in the clear on the blockchain. Transactions are encrypted and the correctness of contract execution can be verified by zero-knowledge proofs.

Our contract system is not for monetary transactions, but employment contracts do contain sensitive personal information; therefore, it is necessary for contract contents to be confidential to others except the involved parties. In our contract blockchain, the contract contents are not shared over the blockchain; only a hash of the contract is distributed and shared. Instead, contractual parties can validate their contracts for accuracy before the contract chain is updated. In addition, all contracts are encrypted and stored on a cloud server for their future use. Our contract system preserves transactional privacy in a more practical way.

2.2. Privacy-Preserving Encrypted Keyword Search

Our proposed system uses encrypted keyword search technology for finding contracts on the server without decrypting the data. To protect data privacy, sensitive data must be encrypted before outsourcing to a cloud server, and encrypted data search is inevitable. Since Song et al. [15] introduced searchable encryption, researchers have conducted numerous efforts to develop searchable encryption. Boneh et al. [16,17] introduced public key encryption with keyword search (PEKS) based on identity-based encryption [18] using bilinear Diffie-Hellman and trapdoor permutation. The shortcoming of PEKS is that the server has to scan all encrypted keywords in sequence to find a matched document.

To overcome the inefficiency of the sequential scan of PEKS, investigators suggested index-based keyword searching. An index contains a list of keywords mapped to their documents, and Goh [19] introduced secure index-based search functionality, proposing two secure index structures, IND-CKA and Z-IDX, based on Bloom filters. Liu et al. [20] proposed searchable encryption with multiple data sources, specifically, multiple indices from different sources. Wang et al. [21,22] introduced similarity scores based on the keyword frequency in a document so that searched results can be ranked by the similarity score, and Cao et al. [23] proposed multi-keyword ranked searching based on secure inner product computation; this process returns documents in the order of their relevance to these keywords. Rajan et al. [24] proposed a dynamic multi-keyword search algorithm that searches documents with multiple keywords sorted by keyword frequency, and Yin et al. [25] suggested more efficient multi-keyword conjunctive querying using a tree-based index. Wu et al. [26] proposed verifiable public key encryption with searching using an inverted encryption index without using a query trapdoor so that file receivers can verify the correctness and completeness of search results. Cao et al. [27] proposed attribute-based encryption with enhanced access control, which provides a secure index to resist keyword guessing attacks from access and search patterns.

To prevent the malicious behavior of a search server, Hu et al. [28] proposed an encrypted search model using blockchain, in which every single search process is managed as a smart contract and recorded on the blockchain; thus, it guarantees the fairness of the search process. Jiang et al. [29] provided an improved blockchain-based encrypted search model allowing multi-keyword search.

Our proposed encrypted keyword search scheme is based on Gentry's homomorphic encryption [30,31]. Homomorphic encryption allows calculations on encrypted data without decrypting it; that is, the result of the computation performed on encrypted data is the same as the result of the computation for the original data. The security of Gentry's scheme is based on the hardness of the approximate integer gcd [32]; in other words, for given a list of integers that are near multiples of a hidden integer, it finds that hidden integer. Although homomorphic encryption is less efficient than symmetric searchable encryption, we adopted homomorphic encryption for our encrypted search for the following reasons: to generate new search tokens for every single search request, to make only the contract owner generate a valid search token for his or her contracts and not to expose the original keyword information to others including even the cloud server. Accordingly,

our proposed search mechanism is secure against keyword guessing attacks and provides accurate search results against keyword synonyms by also considering contract ownership. We also propose a file index for efficient keyword searches.

3. Materials and Methods for Employment Contracts Allowing Encrypted Keyword Searches

3.1. Cryptographic Prerequisite

We briefly review the approximate GCD problem (A-GCD) [32] and the homomorphic hash function [23], which provide the basic security of the proposed scheme.

A. A-GCD

Let λ be a security parameter, $\rho = \lambda$, $\eta = O(\lambda^2)$, and $\gamma = O(\lambda^5)$. The (ρ, η, γ) A-GCD is defined as follows:

Given polynomially many samples from $D_{\gamma,\rho}(p)$ for a randomly chosen η -bit odd integer p, output x, where

$$D_{\gamma,\rho}(p) = \{Choose\ q \Leftarrow \mathbb{Z} \cap [0, 2^{\gamma}/p), r \Leftarrow \mathbb{Z} \cap (-2^{\rho}, 2^{\rho}) :\ Output\ x = p \cdot q + r\}$$

B. Homomorphic hash function

Let G be a multiplicative cyclic group of order p and (g_1, g_2, ... , g_n) be generators. For a vector $b = (b_1, b_2, \ldots, b_n)$, its homomorphic function is defined as $H(b) = \prod_{i=1}^{n} g_i{}^{b_i}$. Then $H(b)$ satisfies the following properties:

- Homomorphic: For any two vectors b_1, b_2 and random integers r_1, r_2 then $H(r_1 b_1 + r_2 b_2) = H(b_1)^{r_1} H(b_2)^{r_2}$.
- Collision Free: For any polynomial time algorithm, it is hard to find b_1, b_2, b_3, r_1 and r_2, which satisfies $H(b_3) = H(b_1)^{r_1} H(b_2)^{r_2}$ but $b_3 \neq r_1 b_1 + r_2 b_2$.

3.2. System Model, Assumptions, and Notation

3.2.1. System Model

We propose an online labor contract management system that provides functionality for contract creation, signing, storage, and management. The system consists of users, employment contract software called *PTEC*, a cloud server, a trusted key generation center (KGC), and a contract blockchain denoted as *C-Chain*. Figure 1 shows the system configuration and the main operations between the components mentioned above. The main roles of each component are as follows:

- User: Users are divided into workers or employers. A pair of one worker and one employer signs an employment contract using *PTEC* over the P2P networks.
- Employment contract software: *PTEC* is a contract application that provides all the functions supported by the proposed system, including contract creation, signing, uploading, searching, and reading.
- Cloud server: It stores encrypted contracts and manages access to the contracts. With a user's contract request, it finds and responds the corresponding contract if the request is valid.
- KGC: A trusted key generation center that communicates with the user's *PTEC* and the cloud server. It generates secret identifiers for registered users and also provides homomorphic encryption keys to the server for the proposed encrypted keyword search.
- *C-Chain*: A blockchain for contract recording to prevent contract forgery.

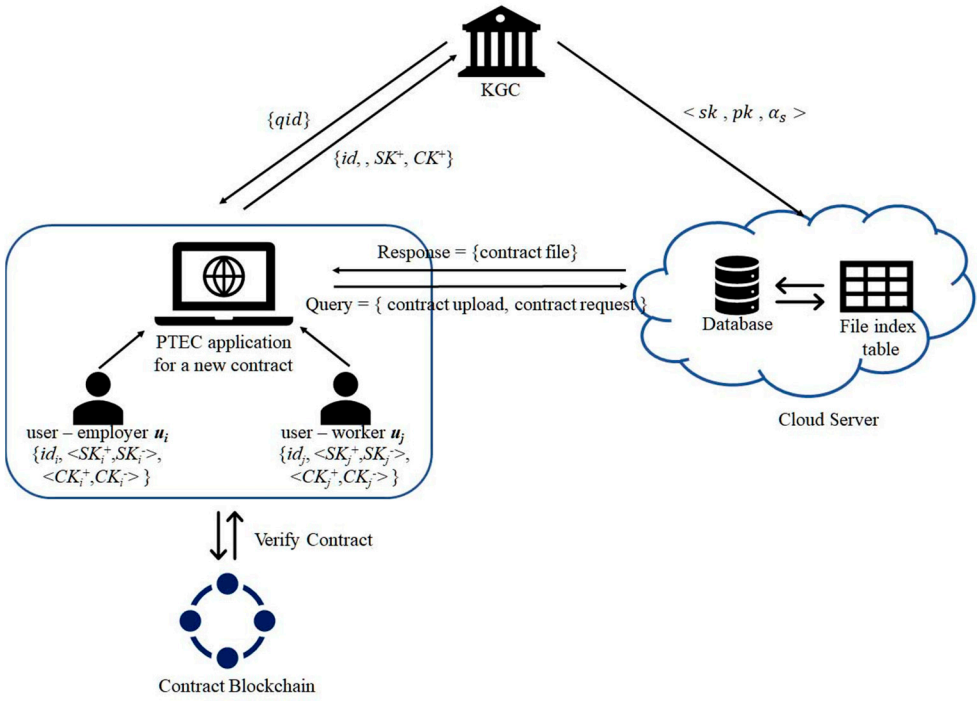

Figure 1. System configuration and operations.

The KGC first generates system parameters and configures the cloud server, and users can use our system just by installing *PTEC*. Users register with the KGC by sending their individual identifier and public keys for signature and encryption and receive a secret identifier from the KGC. For the registered users, a pair of a worker and an employer can create a new contract using *PTEC*; both parties complete the electronic contract and generate their digital signatures, and the final contract is added to *C-Chain* if both signatures are valid. For permanent cloud server storage and access, the names of the two contractual parties are used to generate keywords and those are assigned to the contract. The contracts are encrypted with the user's encryption public key and the keywords are encrypted with the cloud server's public key using Gentry's homomorphic encryption. Finally, the encrypted contract is uploaded to the cloud server along with the encrypted keywords, which users can then use to request their contracts. In our system, each user generates a new encrypted keyword for each single contract access request. The cloud server selects contracts by matching contract keywords to a user's encrypted keyword and access token.

3.2.2. Assumptions, Threat Model, and Security Requirements

We assume the following for the practical use of the proposed system:

- The KGC is secure and trusted.
- The cloud server carries out the prescribed protocol and keeps its secret keys secure.
- Authorized users use the proposed contract system using their own *PTEC*s, and they never expose their secret identifiers or private keys for encryptions and signatures.
- *PTEC* is secure and reliable and can manage all parameters used in the proposed system. Users cannot access other users' encrypted information.
- *C-Chain* can be only updated by *PTEC*s of registered users.
- The communication with the KGC is secure. All KGC data are encrypted with a systemically predefined master key hidden in *PTEC*.

The above assumptions correspond to the basic requirements for implementing the proposed system, so they can be solved with a secure implementation. We also define a threat model for our system as follows:

- Attackers can create contract request queries for other users with predicted keywords.

- Attackers can obtain all information communicated over the network between users and the cloud server, including known encrypted keywords and actual keyword pairs.
- The cloud server can only obtain encrypted contracts and encrypted keywords but can search all data stored on the server.

For secure and reliable use, we designed the proposed system to satisfy the following security requirements:

- Contract confidentiality: Only the owner of a contract can read the contract.
- Contract unforgeability: It is impossible to forge signed contracts.
- Undeniability of contract signing: A contract owner cannot repudiate that he or she signed the contract.
- Contract verifiability: Users can always obtain their original contracts from the cloud server and verify the validity of the contract using the *C-Chain*.
- Contract accessibility: Only registered users can upload contracts to the cloud server and also generate valid request queries for their searches, and users can only access their own contracts.
- Authorized contract search: Only a searcher who knows the secret key p can retrieve a user's contract by performing the proposed encrypted keyword search in the cloud server.
- Contractor anonymity: It is impossible to identify the signer of the contract with the encrypted keywords.

3.2.3. Notation

Before we describe the specific protocols we used to develop our system, we briefly summarize all notations used in this paper. First, for our proposed system, we used the short version of a standard employment contract form, as shown in Table 1.

Table 1. Standard Employment Contract—Short Form.

Contractor	Contract Specification
Employer	Name Company registration number Employment period Working place Main tasks Working hours Break hours Working days (day of the week) Break days (day of the week) Wage (amount/bonus/extra pay/payment date/payment method/etc.) Social insurance Contract date Contact information (company name/phone number/address/employer name/etc.)
Employee	Name Social identifier Contact information (company name/phone number/address/etc.)

Table 2 shows main parameters and notations used in the rest of the paper.

Table 2. Notation.

Division	Notation	Description
KGC	SK_{KGC}	KGC's secret key for user registration and verification
user	u_i	The i-th user for a user set $U = \{ u_1, u_2, \ldots , u_n\}$. A user can be either a worker or an employer.
	qid_i	u_i's secret identifier
	$<SK_i^+, SK_i^->$	u_i's public key pair for digital signature generation and verification
	$<CK_i^+, CK_i^->$	u_i's public key pair for contract encryption and decryption
Digital signature	$S(K^-, H_0(M))$	EC [33] digital signature generation algorithm for a message M with a private key K^-
	$V(K^+, Sig)$	EC digital signature verification algorithm for a signature Sig with a public key K^+
	Sig_i	A digital signature of u_i
Encryption & Decryption	$PKE(K^+, M)$	ECIES public key encryption algorithm for a plaintext M with a public key K^+
	$PKD(K^-, C)$	ECIES public key decryption algorithm for a ciphertext C with a private key K^-
	$E(S, M)$, $D(S, C)$	AES encryption and decryption algorithms with a symmetric key S
Hash function	H_0 (M)	A hash function (SHA-256) for a message M, which generates a hash string of 256 bits.
	$H_1(b = (b_1, b_2, \ldots , b_n))$	A homomorphic hash function for a vector b
	$H_x(M)$	A hash function generating a string of x bits length for a message M. H_x: $\{0,1\}^* \rightarrow \{0,1\}^x$.
Contract	$C_{i,j}^k$	A temporary contract generated at the k-th step between an employer u_i and a worker u_j
	$C_{i,j}$	A final contract file between an employer u_i and a worker u_j (a json file)
Blockchain	$T_{i,j}$	A transaction for $C_{i,j}$
	C-Chain	Blockchain for contract transactions
Keyword-based search query	w_i	u_i's keyword for contract search
	KT_i^t	A keyword token for a keyword w_i generated at time t
	AT_i^t	An access token for a user u_i generated at time t
Cloud server	KT	A set of keyword tokens stored on the cloud server $KT = \{kt_1, kt_2, kt_3, \ldots , kt_n\}$
	$Fid_{i,j}^t$	The identifier of a contract file that is generated between u_i and u_j at time t

3.3. Employment Contracts Allowing Encrypted Keyword Searches

The main operations of our system are divided into four modules: (1) system setup by the KGC, (2) user registration, (3) blockchain-based employment contract creation using *PTEC*, and (4) contract management by the cloud server. In the next subsections, we describe the concrete protocols for each module in detail.

3.3.1. Setup by KGC

The KGC sets its secret key SK_{KGC} and selects all kinds of cryptographic protocols necessary for our system, including symmetric key encryption/decryption algorithms, public key encryption/decryption algorithms, digital signature algorithms, and the homomorphic encryption algorithm. As described in Table 2, our system uses an elliptic curve digital signature scheme, an elliptic curve integrated encryption scheme (ECIES) for public key encryption and decryption, and an advanced encryption standard algorithm for symmetric encryption and decryption. In addition, the KGC sets security parameters required for the proposed encrypted keyword search algorithm. The parameters are determined as follows:

Let l be the bit length of a keyword hash, and let q be the bit length of a user's secret identifier; r denotes the bit length of noise. The values of l, q, and r are systemically predefined. In our experiment, those values are set to $l = 160$, $r = 60$, and $q = 100$.

For a given l, r, and q, a security parameter λ is determined as an integer satisfying $\lambda > \max(l, r + q)$. For example, we set security parameter λ as 164 in our experiment, and the parameters $<\rho, \eta, \gamma, \tau>$ for the homomorphic encryption are determined as $\rho = \lambda$, $\eta = O(\lambda^2)$, $\gamma = O(\lambda^5)$, $\tau = \gamma + \lambda$. The private key of the cloud server is p, and p is a prime such as $p \geq 2^{10 \cdot \lambda}$. Another secret α_s for the cloud server is a prime that satisfies the following condition:

$$(p \bmod \alpha_s) \neq 0 \text{ where } 2^l + 2^{q+r} < \alpha_s < p.$$

Here, $*[z]_p = z \bmod p$ has values within $-p/2 < [z]_p \leq p/2$. The public key of the cloud server is $pk = \{x_0, x_1, \ldots, x_\tau\}$, and the $\tau + 1$ public keys $x_0, x_1, \ldots, x_\tau$ are generated by $D_{\gamma,\rho}(p)$ that is described in Section 3. Here, x_0 is the largest odd number among the public keys and satisfies the following two conditions: $x_0 \bmod p = \alpha_s$ and $x_0 \bmod \alpha_s \neq 0$.

3.3.2. User Registration

The main task of registering users with the KGC is to assign a secret identifier to a user. A new user u_i first creates an identifier id_i, a key pair $<SK_i^+, SK_i^->$ for digital signature generation and verification, and another key pair $<CK_i^+, CK_i^->$ for contract encryption and decryption. u_i sends $< id_i, SK_i^+, CK_i^+>$ to the KGC. The KGC creates a secret identifier for id_i using $KeyGen(id_i)$ algorithm, defined as follows:

KeyGen(id_i): For an input id_i
1. it outputs a secret identifier: $qid_i = H_q\ (id_i \oplus\ SK_{KGC})$.

The algorithm *VerifyUser*(id_i, *auth_code*, *timestamp*) validates the legitimacy of a user u_i. The user generates an authentication code $auth\_code = H_0(id_i \mid qid_i \mid timestamp)$ for id_i, qid_i, and current *timestamp* and sends $<id_i$, *auth_code*, *timestamp* $>$ to the KGC. The *VerifyUser*() algorithm works as follows:

VerifyUser(id_i, *auth_code*, *timestamp*): For given id_i, *auth_code and timestamp*
1. it computes $qid = H_q\ (id_i \oplus\ SK_{KGC})$, $AC = H_0(idi \mid qid \mid timestamp)$, 2. if *auth_code* = *AC* then it outputs "valid." 3. else it outputs "fail."

3.3.3. Blockchain-Based Employment Contract Creation

A pair of a worker and an employer can create a new employment contract using *PTEC* in three steps: (1) contract creation, (2) *C-Chain* update, and (3) contract upload to the cloud server.

A. Contract Creation

Let an employer be u_i, and let a worker be u_j. The contract is created interactively between the two participants in five steps as shown in Figure 2. Each temporary contract generated in each step is encrypted with a recipient's public key and temporarily stored on the cloud server. Once the contract is created, the temporary contract files in progress are removed from the cloud server.

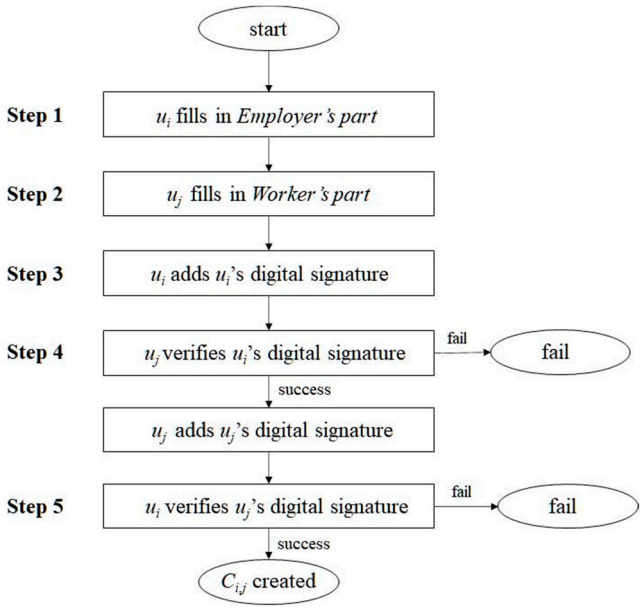

Figure 2. Contract creation steps.

Step 1: u_i creates a new contract denoted as $C_{i,j}{}^1$. u_i fills out the employer's part in the online contract form. $C_{i,j}{}^1$ is encrypted with u_j's public key and stored temporarily on the cloud server.

Step 2: u_j decrypts $C_{i,j}{}^1$ and fills out the worker's part, which is denoted as $C_{i,j}{}^2$. $C_{i,j}{}^2$ is also encrypted with u_i's public key. The encrypted version $EC_{i,j}{}^2 = PKE(CK_i{}^+, C_{i,j}{}^2)$ is stored on the cloud server.

Step 3: u_i decrypts $C_{i,j}{}^2$ and attaches its signature for $C_{i,j}{}^2$ such as $Sig_i = S(SK_i{}^-, H_0(C_{i,j}{}^2))$. That is, $C_{i,j}{}^3$ is u_i's signed contract, as in $C_{i,j}{}^3 = \{C_{i,j}{}^2 \mid Sig_i = S(SK_i{}^-, H_0(C_{i,j}{}^2))\}$. $C_{i,j}{}^3$ is also encrypted with u_j's public key (such as $EC_{i,j}{}^3 = PKE(CK_j{}^+, C_{i,j}{}^3)$) and stored on the cloud server.

Step 4: u_j decrypts $C_{i,j}{}^3$ and validates u_i's signature. If the signature is valid, u_j attaches its signature to $C_{i,j}{}^3$, which is denoted as $C_{i,j}{}^4$. Finally, $C_{i,j}{}^4 = \{C_{i,j}{}^3 \mid Sig_j = S(SK_j{}^-, H_0(C_{i,j}{}^3))\}$, and the encrypted $EC_{i,j}{}^4 = PKE(CK_i{}^+, C_{i,j}{}^4)$ is stored on the cloud server. If the signature is invalid, the contract is stopped at this stage.

Step 5: u_i decrypts $EC_{i,j}{}^4$ and finally verifies the validity of u_j's signature, and if the signature is valid, the contract creation is completed. Consequently, the final contract $C_{i,j}$ between u_i and u_j is defined as follows:

$$C_{i,j} = C_{i,j}{}^4 = \{C_{i,j}{}^2 \mid Sig_i = S(SK_i{}^-, H_0(C_{i,j}{}^2)) \mid Sig_j = S(SK_j{}^-, H_0(C_{i,j}{}^3))\}$$

Otherwise, the contract creation fails.

B. *C-Chain* update

Once a contract is successfully created, *PTEC* of u_i generates a new transaction $T_{i,j}$ for the contract $C_{i,j}$. $T_{i,j}$ is defined as follows:

$$T_{i,j} = \{TID \mid C_{i,j}\} = \{TID \mid Sig_i \mid Sig_j \mid C_{i,j}{}^2\}, \text{ where } TID = H_0(C_{i,j}).$$

PTEC broadcasts only *TID* over the network, and *TID* can be validated by the contractual parties associated with the transaction. If *TID* is not correct, the transaction is rejected by the signer of the contract corresponding to *TID*. At the end of a block interval, for every valid *TID*, every *PTEC* that created a new transaction in the block interval performs a proof of work on the transactions collected in the block interval in order to add the transaction

to the *C-Chain*. After mining a nonce that satisfies a predefined difficulty in our system, it broadcasts a new block information as follows:

$$Block = PreviousBlockHash \mid timestamp \mid nonce \mid NewRootHash,$$

where *NewRootHash* is a root hash of the Merkle hash tree made up of all *TID*s. The *Block* is verified by all peers on the network, including verifying that the hash of *Block* satisfies the difficulty of the system. If more than half of peers agree to the *Block*, then, finally, *C-Chain* is updated as follows and broadcasted over the network:

$$C\text{-}Chain = H_0(PreviousBlockHash \mid timestamp \mid nonce \mid NewRootHash)$$

PTEC keeps *C-Chain* updated. For practical use, the latest *C-Chain* is also stored at the cloud server. Thus, *PTEC* can always get the latest *C-Chain* from the cloud server.

C. Contract Upload

Once *C-Chain* for $C_{i,j}$ is updated, $C_{i,j}$ is encrypted with each signer's public key separately and uploaded to the cloud server along with encrypted keywords for permanent storage and use. Because the contract is encrypted with each signer's public key, only the contract signer can decrypt the contract. Furthermore, the keywords are homomorphically encrypted with the cloud server's public key so that each contract is always searchable by the cloud server with the keywords. The *PTEC* of each u_i related to $C_{i,j}$ performs the following steps to upload each user's contract to the cloud server.

Step 1: Contract encryption

- *PTEC* generates an encrypted contract $EC_i = PKE(CK_i^+, C_{i,j})$.

Step 2: keyword and access token generation

For each contract, a pair consisting of an encrypted keyword token and a contract access token is required for the cloud server to retrieve u_i's contract. The contract access token is required to prove the ownership of the contract. Let u_i's identifier, keyword and secret identifier be id_i, w_i and qid_i, respectively.

- *PTEC* generates a pair of keyword token KT_i and access token AT_i using *GenKeyword-Token*(id_i, w_i, qid_i, pk).

The *GenKeywordToken*() algorithm generates homomorphically encrypted keyword tokens for a keyword w, a secret identifier qid, and the cloud server's public key pk. It outputs two types of tokens denoted as *KT* and *AT*; *KT* is an encrypted keyword, and *AT* is an access token to show the ownership of the contract. *AT* distinguishes the signer from others who might have the same keyword. Note that only registered users should be able to store contracts in the cloud server; thus, each user must first be authenticated by the KGC before the token is generated. For an authorized user, the KGC responds with a session key that is used to generate the token. In other words, the session key is required to ensure that only a registered user can upload a contract to the cloud server. A session key is generated by the following *GetSessionKey*() algorithm:

GetSessionKey(id, qid):

1. it generates $auth\_code = H_0(id \mid qid \mid timestamp)$ with a *timestamp* for the current time,
2. it generates a session key request SK_RQST = "session key request | id | $auth\_code$ | $timestamp$", and
3. it sends SK_RQST to the KGC.
4. the KGC performs *VerifyUser(id, auth_code, timestamp)*.
5. if valid, the KGC
6. chooses α_u that is a prime satisfying $2^l + 2^{q+r} < \alpha_u$ and (p mod α_u) $\neq 0$,
7. generates $auth\_code_\alpha = H_0(\alpha_u \mid \alpha_s \mid timestamp)$, and
8. sends <"verified", α_u, $auth\_code_\alpha$> to u_i.
9. else the KGC sends a "failed" message to u_i.
10. it outputs the KGC's response.

The *GenKeywordToken*() algorithm is defined as follows:

GenKeywordToken(id, w, qid, pk):

1. it gets a session key α_u and its authentication code $auth\_code_\alpha$ using *GetSessionKey*(*id*, *qid*),
2. it chooses a random subset $S \subseteq \{1, 2, \ldots, \tau\}$ from the public key $pk = \{x_0, x_1, \ldots, x_\tau\}$ of the cloud server,
3. it chooses a random integer R, and
4. it computes $<C_1, C_2, C_3>$ as follows:
5. $C_1 = [H_l(w) + R \cdot qid + \alpha_u \sum_{i \in S} x_i]_{x_0}$
6. $C_2 = H_1(R \cdot qid)$
7. $C_3 = [qid + R \cdot qid + \alpha_u \sum_{i \in S} x_i]_{x_0}$
8. it outputs $<KT = (C_1, C_2), AT = (C_2, C_3), \alpha_u, auth\_code_\alpha, timestamp>$.

Step 3: Contract upload

- *PTEC* sends an encrypted contract upload request UP_RQST_i = <"contract upload", $TID, EC_i, KT_i, AT_i, \alpha_u, auth\_code_\alpha, timestamp$> to the cloud server.

D. Contract Request

After the contract upload has been completed, u_i can request its contract from the cloud server at any time. For the contract request, u_i's *PTEC* generates a new pair of KT_i and AT_i using *GenKeywordToken*(id_i, w_i, qid_i, pk). Finally, *PTEC* sends a contract request query CRT_RQST = <"contract request", $KT_i, AT_i, \alpha_u, auth\_code_\alpha, timestamp$> to the cloud server. We describe how the cloud server stores the uploaded contract and responds to the user's contract request in the next section.

3.3.4. Contract Management and Encrypted Keyword Searches by Cloud Server

The primary role of the cloud server is to securely store users' contracts and deliver correct contracts to user requests. As mentioned before, the contracts stored on the cloud server are confidential and anonymous to the server, but the server does need to be able to retrieve the requested contracts accurately and efficiently. To achieve this, the cloud server slightly modifies the uploaded tokens and then stores the contracts in its database (DB) server along with the modified tokens. We describe how the server modifies the uploaded tokens and indexes contract files to increase search efficiency.

A. Contract Storage and File Index Table Update

For a single contract $C_{i,j}$, each contractual party uploads its encrypted contract and tokens. For each upload request *UP_RQST*, the cloud server first verifies if the request was created by an authorized user. For α_u, $auth\_code_\alpha$ and *timestamp* given in *UP_RQST*, the server computes $AC = H_0(\alpha_u \mid \alpha_s \mid timestamp)$ with its secret α_s. If $AC = auth\_code_\alpha$, then the request is valid. Because the user possesses a valid $auth\_code_\alpha$ that can be only generated by the KGC knowing the secret α_s, the server can be convinced that the KGC has authenticated the user.

If the upload request is valid, the cloud server assigns a single file identifier to all encrypted contracts and tokens created by both contractual parties u_i and u_j. In other words, a single file identifier is assigned to all encrypted contracts and tokens with the same *TID*. The file identifier, which is a sequentially increasing number, is automatically generated by the server. Let $Fid^t_{i,j}$ be a file identifier for a set of files related to $C_{i,j}$ uploaded at time t. Consequently, $Fid^t_{i,j}$ indicates a set of $< EC_i, EC_j, KT_i, KT_j, AT_i, AT_j >$.

Here, the tokens are generated with individual α_u randomly assigned to each user by the KGC, which means that the server needs to store α_u additionally for each contract. In order to eliminate this inefficiency, the cloud server re-encrypts the given tokens by replacing α_u with its private secret α_s. For each set of $KT = (C_1, C_2)$ and $AT = (C_2, C_3)$, C_1 and C_3 are recomputed as follows:

1. $P_1 = C_1 \bmod p \bmod \alpha_u$, $P_3 = C_3 \bmod p \bmod \alpha_u$;
2. $C_1{}' = [P_1 + \alpha_s \sum_{i \in S} x_i]_{x_0}$, $C_3{}' = [P_3 + \alpha_s \sum_{i \in S} x_i]_{x_0}$.

Hence, the cloud server can retrieve each user's contract with the server's secret α_s without storing individual α_u. Then, the server stores the encrypted contracts and the modified tokens with its file identifier.

The cloud server maintains a file index table for efficient contract search, as shown in Table 3. The table consists of two attributes: keyword token and bit string; for each keyword token in the table, the bit string represents whether each contract contains the keyword or not. The table contains only distinct keyword tokens. For example, if two different keyword tokens KT_1 and KT_2 indicate to the same keyword w, KT_1 and KT_2 are homomorphic to each other and one of them is added to the table; that is, a previously uploaded keyword token is added to the table. Suppose that $KT = \{KT_1, KT_2, \ldots, KT_j\}$ is the current set of distinct keyword tokens uploaded to the cloud server. As shown in Table 3, the bit string of each keyword token KT_i is determined as follows: for all file identifiers $Fid^t_{i,j} = \{1, \ldots, n\}$, if the keyword token of the contract indicated by each file identifier is homomorphic to KT_i, then 1 is assigned; otherwise, 0 is assigned.

Table 3. File Index Table.

(1) the Way of Determining Bit String of Each Keyword Token

KI \ *FID*	$Fid^1_{1,2}$=1	$Fid^2_{1,3}$=2	$Fid^3_{1,2}$=3	⋯	$Fid^t_{i,j}$=n	**Bit String**
KT_1	1	1	1	…	0	1110 … 0
KT_2	1	0	1	…	0	1011 … 0
KT_3	0	1	0	…	0	0101 … 0
…	…	…	…	…	…	…
KT_j	0	0	0	…	1	0000 … 1

(2) file Index Table

KT	**Bit String**
KT_1	1110 … 0
KT_2	1011 … 0
KT_3	0101 … 0
…	
KT_j	0000 … 1

Whenever a new contract set is uploaded, the table is updated. First, the cloud server compares the given keyword token KT_k with all the KT_i in KT. If there is a KT_j such that the keyword w_j represented by KT_j is the same as the keyword w_j represented by KT_k, then the bit strings of all keyword tokens in the table are updated. Otherwise, a new KT_k is added to the table and all the bit strings are updated.

B. Keyword and Access Token Comparison

The key operation of the cloud server is to determine whether any two tokens are homomorphic or not. We propose two test algorithms to determine the homomorphism of two tokens. Suppose that KT_i is a keyword token for a keyword w_i and KT_j is a token for w_j. The two different keyword tokens KT_i and KT_j are homomorphic if $w_i = w_j$. The algorithm $Test\_KT(KT_i, KT_j)$ determines whether two tokens KT_i and KT_j are homomorphic or not.

$Test\_KT(KT_i, KT_j)$: Suppose that $KT_i = <C_{i1}, C_{i2}>$ and $KT_j = <C_{j1}, C_{j2}>$. It computes the following Equation (1). If the result is 1, then the two tokens are homomorphic. That is, $w_i = w_j$. Otherwise, $w_i \neq w_j$.

$$\frac{H_1(C_{i1} mod\ p\ mod\ \alpha_s) \times C_{j2}}{H_1(C_{j1} mod\ p\ mod\ \alpha_s) \times C_{i2}} \quad (1)$$

Similarly, the algorithm $Test\_AT(AT_i, AT_j)$ determines whether two tokens AT_i and AT_j are homomorphic or not. Suppose that AT_i is an access token for a secret identifier qid_i and AT_j is a token for a secret qid_j. AT_i and AT_j are homomorphic if $qid_i = qid_j$.

Test_AT(AT_i, AT_j): Suppose that AT_i = <C_{i2}, C_{i3}> and KT_j = <C_{j2}, C_{j3}>. It computes the following Equation (2). If the result is 1, then $qid_i = qid_j$. Otherwise, $qid_i \neq qid_j$.

$$\frac{H_1(C_{i3} mod\ p\ mod\ \alpha_s) \times C_{j2}}{H_1(C_{j3} mod\ p\ mod\ \alpha_s) \times C_{i2}} \quad (2)$$

C. Contract Search

Users can always request their contracts to the cloud server. For a contract request, a user u_i creates a contract request query *CRT_RQST* = < "contract request", KT_i, AT_i, α_u, $auth\_code_\alpha$, *timestamp*> using *GenKeywordToken*(id_i, w_i, qid_i, *pk*). The cloud server first verifies the validity of the request using $auth\_code_\alpha$ and the server's secret α_s. If the request is authorized, then the server performs a keyword-based contract search. It finds a keyword token in the file index table that is homomorphic to KT_i. In other words, it finds a token KT_j such that *Test_KT*(KT_i, KT_j) = 1 by doing *Test_TK*(KT_i, KT_j) for all KT_j in *KT*. If there is no such KT_j, the search fails. If the search succeeded, the bit string of KT_j represents the file identifiers of the contracts containing the same keyword as the requested keyword. For all file identifiers such that the bit value of each file identifier is 1, it performs *Test_AT*(AT_i, AT_j) for all access tokens AT_j attached to the contracts of the selected file identifiers. Finally, it finds contracts such that *Test_AT*(AT_i, AT_j) = 1 and responds with the contracts to the request.

Additionally, u_i' can make a request with multiple keywords. For example, suppose that a worker u_i wants to search a contract containing both u_i's name w_i and an employer u_j's name w_j. Then, u_i generates two sets of keyword tokens <KT_1, AT_1> and <KT_2, AT_2> for w_i and w_j, respectively. <KT_1, AT_1> is the output by calling *GenKeywordToken*(id_i, w_i, qid_i, *pk*) and <KT_2, AT_2> is the output by calling *GenKeywordToken*(id_i, w_j, qid_i, *pk*). Thus, <KT_1, AT_1> and <KT_2, AT_2> are defined as follows:

$$\begin{aligned}
KT_1 &= \langle C_{11},\ C_{12} \rangle =< [H_l(w_i) + R_1 \cdot qid_i + \alpha_1 \textstyle\sum_{i \in S} x_i]_{x_0},\ H_1(R_1 \cdot qid_i) >,\\
AT_1 &= \langle C_{12},\ C_{13} \rangle =< H_1(R_1 \cdot qid_i),\ [qid_i + R_1 \cdot qid_i + \alpha_1 \textstyle\sum_{i \in S} x_i]_{x_0} >,\\
KT_2 &= \langle C_{21},\ C_{22} \rangle =< \left[H_l(w_j) + R_2 \cdot qid_i + \alpha_2 \textstyle\sum_{i \in S} x_i\right]_{x_0},\ H_1(R_2 \cdot qid_i) >,\\
AT_2 &= \langle C_{22},\ C_{23} \rangle =< H_1(R_2 \cdot qid_i),\ [qid_i + R_2 \cdot qid_i + \alpha_2 \textstyle\sum_{i \in S} x_i]_{x_0} >,
\end{aligned}$$

where a pair of (R_1 and α_1) and a pair of (R_2 and α_2) are random values used in each *GenKeywordToken*() function for w_i and w_j, respectively.

When two sets of keyword tokens are given to the cloud server, it first looks for homomorphic keyword tokens for the given tokens in the file index table. That is, it finds KT_i such that *Test_KT*(KT_1, KT_i) = 1 and KT_j such that *Test_KT*(KT_2, KT_j) = 1. Let the bit string of KT_i be B_i and the bit string of KT_j be B_j. It calculates $B = B_i \cdot B_j$ where $\cdot$ is an AND bit operation. Then, it finds all file identifiers such that the bit value of B is 1. Finally, it finds contracts satisfying *Test_AT*(AT_1, AT_k) = 1 (or *Test_AT*(AT_2, AT_k) = 1). AT_1 and AT_2 are homomorphic access tokens for authenticating the same qid_i, so testing for only one access token is sufficient. Remind that a set of contract files represented by a file identifier $Fid^t_{i,j}$ uploaded at time t is < EC_i, EC_j, KT_i, KT_j, AT_i, AT_j >. There are two different access tokens generated by both signers participated in the contract. *Test_AT*() works for each *AT*; if one of them is 1, then, the contract access is authorized. In this way, the proposed contract search algorithm works for multiple keywords.

4. Theoretical Analysis and Experimental Results

4.1. Correctness of the Encrypted Keyword Search

Only the cloud server that knows the private key p can search contracts with encrypted keywords; we show how the proposed search scheme works correctly. First of all, a user u_i's ciphertexts C_1 and C_3 contained in *KT* and *AT* are modified to C_1' and C_3' by the cloud server. We describe the recalculation for C_1, and it is the same for C_3. C_1 is calculated with

a session key α_u but the cloud server removes α_u and replaces it with the server's secret α_s. We show that the modified $C_1{}'$ outputs the correct search result.

First, C_1 is determined as follows:

$$\begin{aligned}
C_1 &= [H_l(w) + R \cdot qid + \alpha_u \textstyle\sum x_i]_{x_0} \\
&= [H_l(w) + R \cdot qid + \alpha_u(p \cdot q_1 + r_1 + p \cdot q_2 + r_2 + \ldots + p \cdot q_n + r_n)]_{x_0} \\
&= H_l(w) + R \cdot qid + \alpha_u(p \cdot q_1 + r_1 + p \cdot q_2 + r_2 + \ldots + p \cdot q_n + r_n) + k(p \cdot q_0 + r_0) \\
&= H_l(w) + R \cdot qid + \alpha_u(p \cdot q_1 + r_1 + p \cdot q_2 + r_2 + \ldots + p \cdot q_n + r_n) + k(p \cdot q_0) + k \cdot r_0 \\
&= H_l(w) + R \cdot qid + p(\alpha_u \cdot q_1 + \ldots + \alpha_u \cdot q_n + k \cdot q_0) + \alpha_u(r_1 + r_2 + \ldots + r_n) + k \cdot r_0 \\
&= H_l(w) + R \cdot qid + p(\alpha_u \textstyle\sum q_i - k \cdot q_0) + \alpha_u(\sum r_i) + k \cdot r_0
\end{aligned}$$

To remove α_u from C_1, the cloud server firstly computes the following Equation (3).

$$P_1 = C_1 \bmod p \bmod \ = H_l(w) + R{\cdot}qid + k{\cdot}r_0 \tag{3}$$

Then, it computes $C_1{}'$ using α_s and a subset of public keys S selected randomly by the server as follows:

$$\begin{aligned}
C_1' &= [P_1 + \alpha_s \textstyle\sum_{i \in S} x_i]_{x_0} \\
&= [H_l(w) + R \cdot qid + k{\cdot}r_0 + \alpha_s(p \cdot q_1 + r_1 + p \cdot q_2 + r_2 + \ldots + p \cdot q_n + r_n)]_{x_0} \\
&= H_l(w) + R \cdot qid + k{\cdot}r_0 + \alpha_s(p \cdot q_1 + r_1 + p \cdot q_2 + r_2 + \ldots + p \cdot q_n + r_n) + m(p \cdot q_0 + r_0) \\
&= H_l(w) + R \cdot qid + k{\cdot}\alpha_s + \alpha_s(p \cdot q_1 + r_1 + p \cdot q_2 + r_2 + \ldots + p \cdot q_n + r_n) + m(p \cdot q_0 + \alpha_s) \\
&= H_l(w) + R \cdot qid + p(\alpha_s \cdot q_1 + \ldots + \alpha_s \cdot q_n + m \cdot q_0) + \alpha_s(r_1 + r_2 + \ldots + r_n + k + m) \\
&= H_l(w) + R \cdot qid + p(\alpha_s \textstyle\sum q_i + m \cdot q_0) + \alpha_s(\sum r_i + k + m)
\end{aligned}$$

Because $x_0 \bmod p = \alpha_s$, the above equation holds, and the modified encrypted keywords are stored at the server. For a keyword search, two test algorithms *Test_KT*(C_i, C_j) and *Test_AT*(C_i, C_j) need to compute (C_i mod p mod α_s) and (C_j mod p mod α_s) first. For the modified ciphertext $C_1{}'$, the following Equation (4) holds so that the test algorithms work correctly:

$$C_1{}' \bmod p \bmod \alpha_s = H_l(w) + R{\cdot}qid \tag{4}$$

Notice that the initial encrypted keywords stored on the server along with a new contract must be recomputed with the server's secret α_s. Regarding whether or not a contract request query requires the server to recalculate the search ciphertexts of the tokens contained in the request query to find user's contract, the tokens in the query are only required for the homomorphic comparison to the tokens stored on the server.

Suppose that the tokens contained in u_i's new contract request query are <KT^2, AT^2>, where KT^2 = <${C_1}^2$, ${C_2}^2$> and AT^2 = <${C_2}^2$, ${C_3}^2$>. Without loss of generality, ${C_1}^2$ and ${C_3}^2$ are modified to ${C_1}^{2\prime}$ and ${C_3}^{2\prime}$ with α_s by the server to obtain the correct search result. Then, *Test_KT*($C_1{}'$, ${C_1}^{2\prime}$) and *Test_AT*($C_3{}'$, ${C_3}^{2\prime}$) are performed. Here, α_s will be finally removed from ${C_1}^{2\prime}$ and ${C_3}^{2\prime}$ in the testing algorithm, and thus the server does not need to generate ${C_1}^{2\prime}$ and ${C_3}^{2\prime}$ with α_s intentionally before doing the testing algorithms. Therefore, instead of computing ${C_1}^{2\prime}$ with α_s, a modified testing algorithm is used to remove the recalculation. For two search ciphertexts < $C_1{}'$, ${C_1}^2$>, a modified testing algorithm works as follows:

$$\begin{aligned}
\frac{H_1\left(C_1' mod\ p\ mod\ \alpha_s\right) \times C_{j2}}{H_1\left(C_1^2 mod\ p\ mod\ \alpha_u\ mod\ \alpha_s\right) \times C_{i2}} &= \frac{H_1(H_l(\ w_i) + R_i \cdot qid_i) \times H_1\left(qid_j\right)^{R_j}}{H_1\left(H_l\left(w_j\right) + R_j \cdot qid_j\right) \times H_1(qid_i)^{R_i}} \\
&= \frac{H_1(H_l(\ w_i)) \times H_1(qid_i)^{R_i} \times H_1\left(qid_j\right)^{R_j}}{H_1\left(H_l\left(w_j\right)\right) \times H_1\left(qid_j\right)^{R_j} \times H_1(qid_i)^{R_i}} = \frac{H_1(H_l(w_i))}{H_1\left(H_l\left(w_j\right)\right)}
\end{aligned}$$

The process works for <$C_3{}'$, ${C_3}^2$> in the same way. In our simulation, the proposed encrypted search algorithm provides 100% contract search accuracy.

4.2. Security Analysis

Here, we show that our system satisfies the security requirements described in Section 4.1.

(1) Contract unforgeability: Each contract is double signed by both parties. Based on the security of the EC digital signature algorithm, it is impossible to generate a valid signed contract or forge the signed contract without knowing each user's private key. Morever, the hash of the signed contract is publicly recorded on the *C-Chain* so that even the owners of the contract cannot forge their previously signed contracts.

(2) Contract confidentiality: All contracts stored on the cloud server are encrypted with each owner's public key. Depending on the security of the ECIES public key encryption algorithm, it is impossible to decrypt the contract without knowing each owner's private key. Therefore, only contract signers can decrypt and read their own contracts.

(3) Contract accessibility: In order to upload a contract to the server and to generate a contract request query, a user must receive a session key α_u from the KGC to generate encrypted tokens. In this process, the user must show that she or he knows a valid secret identifier *qid* assigned from the KGC in the user registration process. The KGC authenticates the user using the *VerifyUser*() algorithm and issues a session key α_u for a registered user with an authentication code $auth\_code_\alpha$. The code can only be created by the KGC that knows a secret α_s shared with the cloud server. Finally, the server will accept the contract upload or the contract request query if the $auth\_code_\alpha$ a is valid. Therefore, only registered users can receive a valid session key and generate valid search tokens, so they can upload and access their contracts. Moreover, the $auth\_code_\alpha$ contains the current timestamp, so the proposed system prevents replay or reuse of a previously used request query.

Next, as we mentioned before, the keywords are easily predictable. A registered but mischievous user u_m might be able to make a contract request for another user u_k's keyword w_k and succeed in obtaining a valid session key from KGC, but u_m has no choice but to use u_m's own secret identifier qid_m to generate tokens for w_k. Here, the tokens for w_k stored on the cloud server were generated by u_k's secret identifier qid_k, so the server will fail to find a contract that corresponds to u_m's request query. Users cannot access other contracts with only known or predicted keywords; they must also know the keyword owner's secret identifier. Therefore, the proposed encrypted search scheme is secure against the keyword guessing attack. The result is that users can only access their own contracts.

(4) Contract verifiability and undeniability: Each contract is created interactively by both employer and worker and digitally signed by both of them. Then, the hash of the signed contract is publicly recorded on the *C-Chain*. To maintain privacy, no other users can publicly verify the contract content, but contract signers also cannot repudiate the fact that they have signed the contract given that the contract hash containing their signatures is recorded on the *C-Chain*. When a dispute arises, the original contract can be always obtained from the cloud server and validated with the signers' public keys and the *C-Chain*.

(5) Contractor anonymity: The keywords attached to each contract in effect represent the signers of the contract. However, the keywords are first hashed, and then the hashed keywords are homomorphically encrypted and stored on the server along with the encrypted contracts. With the one-way-ness of the cryptographic hash function, it is impossible to know the actual keyword w from the hashed value $H_1(w)$, but computing $H_1(w)$ from w is very easy. The keyword is a common person's name or a known business name, so we can easily obtain all pairs of <w, $H_1(w)$> for well-known names w. Therefore, the keyword hash $H_1(w)$ should not be disclosed from the keyword token. We show that it is also impossible to know the keyword hash $H_1(w)$ from the homomorphically encrypted keyword token $C_1 = [H_l(w) + R \cdot qid + \alpha_s \sum x_i]_{x_0}$ for w. Only the server that knows a secret key p can compute $Key = ((C_1 \bmod p) \bmod \alpha_s) = H_l(w) + R{\cdot}qid$, and the keyword hash $H_l(w)$ is still hidden by a random $R{\cdot}qid$. It is impossible to know $H_l(w)$ without knowing the secret identifier *qid* of the contract owner because R is refreshed randomly in the generation of each keyword token. The server that knows p can test whether any two keywords are homomorphic or not through our keyword test algorithms, but it cannot discover $H_l(w)$ in the keyword search step. Therefore, the actual keyword w is not exposed to any others including the server.

(6) Authorized search by the cloud server: The keyword search is only allowed for the cloud server because only the server knows the secret key p and can perform the encrypted keyword searches correctly. Because we based our keyword encryption on Gentry's homomorphic encryption [32], we prove the security of our model using the security proof of Gentry's encryption. Roughly speaking, if an adversary A performs the encrypted keyword search without knowing p, then we can construct a solver B that can solve the approximate GCD problem using A, but this contradicts the hardness of A-GCD. From Theorem 4.2 in [32], we can conclude the following Lemma 1:

Lemma 1. *Fix the parameters <ρ, η, γ, τ> as in the proposed scheme from Section 4.2 (all polynomial in the security parameter λ). Any adversary A with advantage e on the proposed encrypted keyword search scheme can be converted into an algorithm B for solving <ρ, η, γ>-approximate GCD with success probability at least e/2. The running time of B is polynomial in the running time of A, and in λ and 1/e.*

Proof. We follow Gentry's proof to show how B can recover p with the success probability using A. Let $q_p(z)$ and $r_p(z)$ be the quotient and remainder of z with respect to p, hence $z = q_p(z) \cdot p + r_p(z)$.

Step 1: The solver B constructs $\tau + 1$ public keys $x_0, \ldots, x_\tau$ from $D_{\gamma,\rho}(p)$. It relabels so that x_0 is the largest. It restarts unless x_0 is odd. B outputs a public key $pk = <x_0, \ldots, x_\tau>$.

Step 2: B produces a sequence of integers and attempts to recover p by utilizing A to learn the least-significant bit of the quotients of these integers with respect to p. For this, B uses the following subroutine:

Subroutine Learn-LSB (z, pk):
Input: $z \in [0, 2^\gamma)$ with $\lvert r_p(z) \rvert < 2^\rho$, a public key $pk = <x_0, \ldots, x_\tau>$
Output: The least-significant bit of $q_p(z)$
1. for $j = 1$ do poly(λ)/e do:
2. Choose a random $r_j > 2^l + 2^{q+r}$, a bit $m_j \in \{0, 1\}$, and a random set $S_j \subseteq \{1, \ldots, \tau\}$
3. Set $C_j = \left[z + m_j + r_j \sum x_i\right]_{x_0}$
4. Call A to get a prediction $A_j \leftarrow A(pk, C_j, r_j)$
5. Set $a_j = A_j \bmod 2$
6. Set $b_j = a_j \oplus parity(z) \oplus m_j$
7. Output the majority vote among the b_j's

Line 3 is a valid encryption of the bit $[r_p(z)]_2 \oplus m_j$. Because p is odd, we always have $[q_p(z)]_2 = [r_p(z)]_2 \oplus parity(z)$. Then, Learn-LSB($z$, pk) will return $[q_p(z)]_2$ with overwhelming probability.

Step 3: A is an oracle for the least-significant bit of $q_p(z)$, and recovering p is rather straightforward. Given any two integers $z_1 = q_p(z_1) \cdot p + r_p(z_1)$ and $z_2 = q_p(z_2) \cdot p + r_p(z_2)$ (with $r_p(z_i) << p$), repeatedly apply the following process to them:

Binary GCD:
1. If $z_2 > z_1$ then swap them, $z_1 \leftrightarrow z_2$.
2. Use the oracle to learn the parity bit of both $q_p(z_1)$ and $q_p(z_2)$, denote $b_i = [q_p(z_i)]_2$.
3. If both $q_p(z_i)$ are odd then replace z1 by $z_1 = z_1 - z_2$ and set $b_1 = 0$.
4. For each z_i with $b_i = 0$, replace z_i by $z_i = (z_i - parity(z_i))/2$.
(Note that z_i − parity(z_i) is even, so the new z_i is an integer.)

When $r_p(z_i) << p$, subtracting the parity bity does not change the quotient with respect to p. That is, q_p (z_i − parity(z_i)) = $q_p(z_i)$. When we set $z_i' = (z_i - \text{parity}(z_i))/2$, we get

$$q_p(z_i') = q_p(z_i)/2 \text{ and } r_p(z_i') = (r_p(z_i) - parity(z_i))/2.$$

Hence, after O(γ) iterations we will finally get two integers z_1', z_2' with $z_2' = 0$ and $q_p(z_1')$ being the odd part of GCD($q_p(z_1)$, $q_p(z_1)$).

Step 4: To recover p, the solver B draws a pair of elements z_1^*, z_2^* from $D_{\gamma,\rho}(p)$ and applies the binary GCD algorithm to them. With probability at least 0.6, the odd part

of GCD($q_p(z_1{*})$, $q_p(z_2{*})$) is one, which means that the procedure will output an element $z'' = 1 \cdot p + r$ with $|r| \leq 2^{\rho}$. Lastly B repeats the binary GCD procedure using $z_1 = z_1{*}$ and $z_2 = z''$, and the sequence of parity bits of the $q_p(z_1)$ in all the iterations spell out the binary representation of $q_p(z_1{*})$. Now, B recovers $z_1^* / q_p\left(z_1^*\right)$.

The success probability of B is identical to the proof in [32]. □

4.3. Simulated Performance Analysis

We analyze the computational efficiency of the proposed scheme. Our system consists of two main parts: contract creation using blockchain and the contract search with encrypted keywords. The most time-consuming and crucial operations in creating a new employment contract are performing the proof of work for new contracts in a block and verifying the validity of the contract. In the contract search part, the main operations are keyword token generation, contract upload, and the search keyword token comparison. We simulated the actual processing time for each major task with various values and then analyzed the computational performance of our system with the simulated results. We summarize the system environment of each peer node used in our simulation, the simulation parameters, and their values in Table 4.

Table 4. Simulation Parameters and Values.

System Environment and Parameters	Values
System OS/CPU/RAM	Window 10, 64 bit/i7/8 GB
DB for keyword tokens	Mongo DB
The number of peer nodes	10, 50, 100, 150, 200
The number of names	Top 100 most famous names in S. Korea (the number of names is relatively small in order for the synonym test)
The number of business names	30,206 actual business names selected randomly
The number of distinct keywords	100, 200, 500, 1000, 3000, 5000, 7500

4.3.1. Contract Creation by Users

Firstly, the time efficiency to create a single contract on the user side is analyzed. The time to fill out the contract form depends entirely on the user, so only the computational efficiency of the operations performed by *PTEC* is analyzed. The contract creation consists of five steps as described in Section 3.3.3. Steps 1 through 4 basically include encrypting a temporary contract with the recipient's public key and uploading the temporary contract to the cloud server. Uploading includes both the delivery of the contract to the cloud server and the storage of the contract in the temporary contract DB. In addition, Step 2 through 4 involve downloading the temporary contract, decrypting the contract. Generating a digital signature is added to Steps 3 and 4, and the signature is needed to verify in Steps 4 and 5. Table 5 summarizes the average time of each operation performed by *PTEC*, and Table 6 shows all operations involved in each step and the average time to perform all those operations for each step. Notice that the average time in Table 6 differs from the actual running time of each step because the operations conducted by each user were excluded from the time analysis. As can be seen in Tables 5 and 6, since the amount of time to perform the cryptographic operations and communicate with the cloud server is about 79 ms in total, it can be neglected. Unlike PDF contracts, the user does not need to manually create or attach his or her signature. Just by filling out the contract form, a signed contract that is secure, unforgeable, and verifiable is automatically completed by *PTEC*.

Table 5. The average time of each operation performed by *PTEC*.

Operation	Average Time (ms)
Contract encryption/decryption	2
Uploading a contract to the cloud server	9.5
Downloading a contract from the cloud server	4.25
Digital signature generation	2
Digital signature verification	2

Table 6. The average time of *PTEC*'s operations for each step.

Steps	Operations Performed by *PTEC*	Average Time (ms)
Step 1	Encryption and uploading	11.5
Step 2	Downloading, decryption, encryption, and uploading	17.75
Step 3	Downloading, decryption, signature generation, encryption, and uploading	19.75
Step 4	Downloading, decryption, signature verification, signature generation, encryption, and uploading	21.75
Step 5	Downloading, decryption, and signature verification	8.25

4.3.2. *C-Chain* Update

Every signed contract is recorded on the contract blockchain, *C-Chain*. The most time-consuming operation in *C-Chain* generation is performing the proof of work for a new block containing new contracts; the process requires finding a random nonce that satisfies the predefined blockchain difficulty. Figure 3 shows the random nonce mining time according to various difficulties. The process took less than 1 s at levels under 5 but increased exponentially from level 5 and above, taking about 10 min for level 7.

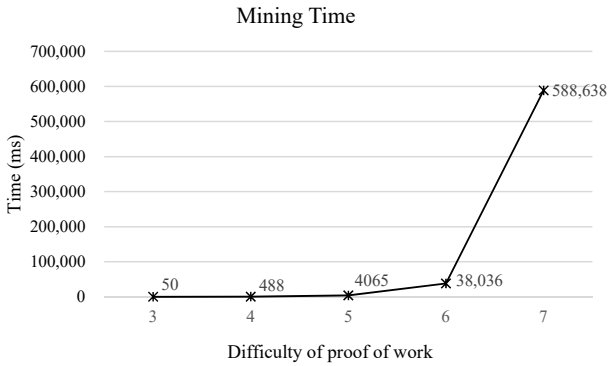

Figure 3. Mining Time.

Once the proof of work has been completed, a block hash for the block is determined that must be validated by users on the network before the *C-Chain* updates. Figure 4 shows the block verification time, which includes computing the block hash of the block with previously broadcasted contract hashes and the given nonce and comparing the computed block hash with the given block hash. This task should be done by peers on the network, and it is verified if more than half of the peers agree to the given block hash. This task works independently with individual users, so the block verification time is affected by the network situation rather than the number of peers in the network. It takes about 3.78 s on average.

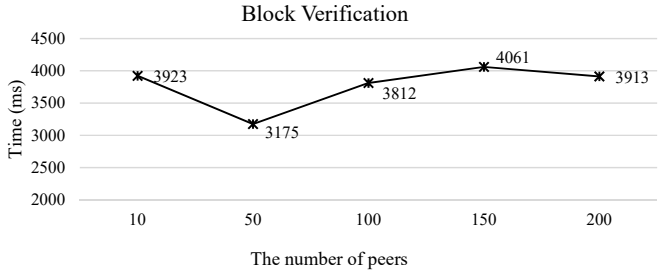

Figure 4. Block Verification Time.

4.3.3. Contract Search with Encrypted Keywords

When the *C-Chain* for a new contract is updated, the contract is uploaded to the cloud server. Before that process, however, searchable keyword and access tokens must be generated, which requires communication with the KGC to get a random α_u. The contract is encrypted, and a set of an encrypted contract and tokens is delivered to the cloud server. Then, the server assigns a file identifier to the contract and stores it. Finally, the server updates the file index table for the given keyword token. To update the table, the server must compare the given keyword token to all tokens in the table to find one that is homomorphic to the given token: That is, an encrypted keyword search occurs. Thus, we analyze the computation time for the main operations: keyword and access token generation, contract encryption and delivery, and keyword search by the server.

Figure 5 shows the keyword and access token generation time. Generating the keyword token includes the KGC's selecting α_u and computing a few hash values for the signer's keyword and secret identifier; this takes about 503 ms on average. The most time-consuming operation is to receive the session key from the KGC. The average time for creating each token with the given session key and the signer's secret identifier is around 6.75 ms. The contract is encrypted with the contract signer's public key and then a set of encrypted contract and keyword and access tokens are delivered to the cloud server.

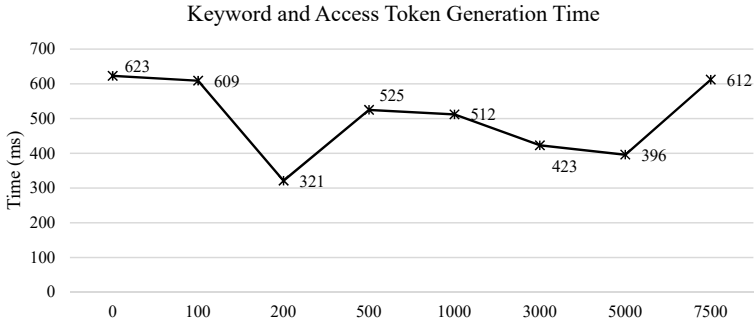

Figure 5. Keyword and access token generation time.

Figure 6 shows the average time for the contract encryption and delivery to the server; it takes about 8 ms. The contract encryption takes about 2 ms on average and the average delivery time to the cloud server is about 6.12 ms. The token generation, contract encryption, and delivery are performed individually by each user, so the processes can take place in real time. In contrast, updating the file index table takes as long as the encrypted keyword search time, which is heavily affected by the size of the table; that is, table updating time is proportional to the number of keyword tokens in the table. Figure 7 shows search times by number of tokens in the table. The homomorphic test for a pair of search tokens takes about 1.13 ms. Because the search is performed sequentially against the tokens in the file index table, the search time is also closely affected by the locations of the searched keywords in the table, and the figure shows three different average search times for different keyword tokens locations. It took less than 1 s for 500 keyword tokens, but for more than 500, the search time difference increased depending on the tokens' locations in the file index table. For keywords at the beginning of the table, the best-case keyword

search, it took about 500 ms until 5000 tokens and 1.34 for 7500 tokens. For keywords in the middle of the table, reflecting average search time, it took 602 ms for 1000 tokens, 2 s for 3000 tokens, and 4.8 s for 7500 tokens. Lastly, for keywords not in the file index table, which shows the worst search time, it took 2.5 s for 1000 tokens, 6.6 s for 5000 tokens, and almost 8 s for 7500 tokens. In terms of practical use of our system, 8 s is a long time for a keyword search. The search time inevitably increases proportional to the number of keyword tokens in the table, but actual running time can vary depending on how the cloud server is implemented. We expect that it can be further improved by allowing parallel search using a distributed DB or a search engine-enhanced DB.

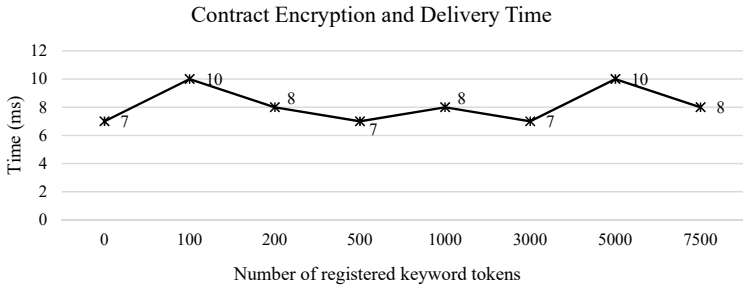

Figure 6. Contract encryption and delivery time.

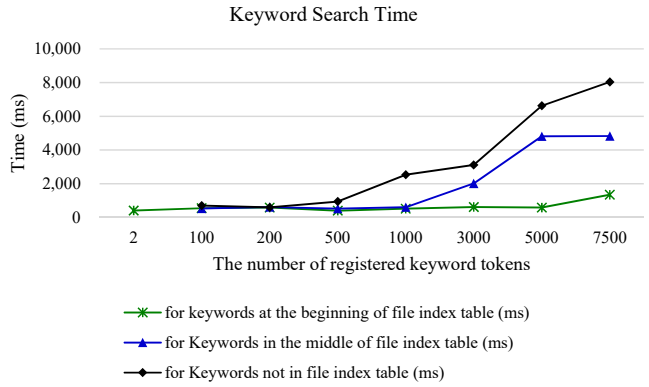

Figure 7. Encrypted keyword search time.

Finally, we summarize the total amount of time for completing the entire process, from creating a new contract to uploading the contract to the cloud server. Table 7 shows a summary of the total execution time with 200 peers and 7500 tokens.

Table 7. Summary of the total execution time.

Main Process	Operations	Average Time (ms)
Contract creation by users		79
Blockchain update(with difficulty level 5)	Mining a block hash	4065
	Block hash verification	3913
Contract encryption & search token generation	Search token generation	612
	Contract encryption & delivery	8
Contract storage including keyword search		4826
Total		13,503

5. Discussion and Conclusions

A secure online employment contract system architecture using blockchain technology to build an unforgeable, undeniable but verifiable has been presented. Comparing current digital contracts such as PDF contracts, the proposed electronic contract system has the advantages of strong security, efficient storage and management, and practical

usability for the following reasons: (1) Non-forgery, non-repudiation, and verifiability are the most essential security requirements of the contract system. The proposed contract system uses a variety of cryptographic technologies including public key cryptography, homomorphic encryption, digital signature, and blockchain to ensure all the most essential security requirements of the contract system. (2) All contracts created in the proposed system are stored and managed integrally on the cloud server. Individuals are not burdened with keeping their contracts secure with no risk of losing them. Moreover, users can always access and use their own contracts anytime, anywhere. (3) In this paper, a dedicated software *PTEC* was used to sign a contract, but the proposed contract system can be implemented on any platform such as web platform and mobile; hence, users can conveniently use it in any environment.

The blockchain technology basically requires public verification of transactions by sharing the transactions with all users on the blockchain network to update the blockchain. However, because each contract contains highly sensitive personal information such as social identifier, personal address, and income, sharing the contracts with others for the blockchain generation is not available in the proposed system. Rather than sharing the contract itself, our model shares only the transaction hash of each contract with others on the blockchain network to preserve the transactional privacy of contracts. Because all of contractual parties participate in the proof of work for their contracts to update the blockchain, each contractual party can validate the transaction hash of his or her own contract. Therefore, the proposed blockchain model serves as the ledger of contract creation record while effectively preserving the personal information of contracts.

In addition to the creation of unforgeable and verifiable contracts, the secure management and use of the created contracts must be supported. For this, we also proposed a secure contract management and contract search model using a cloud server. All the signed digital contracts are securely managed on the cloud server for practical and secure use. This means all contracts are encrypted and stored on the cloud server for confidentiality; however, any contract signer should be able to access his or her contract on the cloud server all the time. To achieve this, we provided an enhanced encrypted keyword search protocol to make the cloud server retrieve each user's contract without decrypting the contract or identifying the actual signer of the contract. Each contract is matched with two or three keywords based on the names of contractual parties; because the keywords are name based, they are easily predictable, and keyword synonyms can occur frequently in the proposed system. This means the server has to sift out only a contract requester's contract among the many contracts with the same keyword, because users should be able to access their own contracts only. Thus, to solve this problem, we proposed an encrypted search scheme considering both the contract keywords and the ownership of each contract. The proposed search model firstly selects contracts matching to a user's encrypted keyword, and then, it verifies the ownership of the selected contracts. To increase the search efficiency, a file index table was proposed, as well. In particular, for multiple keywords search, the proposed algorithm can find contracts matching to all of the multiple keywords with a simple AND bit operation on the file indexes of the keyword-based selected contracts. So, the contract search time can be significantly reduced. As a result, the proposed search model exactly retrieves the requester's contracts; at the same time, it is secure against a keyword guessing attack and provides strong search accuracy against the keywords synonyms.

We provided concrete implementation of the proposed system, demonstrated how the proposed system satisfies the proposed security requirements, and simulated the efficiency and practicality of the proposed system under various parameters. Our proposed system showed 100% search accuracy with the most time-consuming operation being the encrypted keyword search. In the simulated results, the process took an average of 4.8 s for 7500 keyword tokens. We need further research on various approaches to reduce the keyword search time.

Author Contributions: Conceptualization, S.Y., S.C., and S.P.; methodology, S.Y, S.C., and S.P.; software, S.Y. and S.C.; validation, S.Y, S.C., and S.P.; formal analysis, S.P.; writing—original draft

preparation, S.Y. and S.C.; writing—review and editing, J.C. and S.P.; supervision, S.P.; All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

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

References

1. CBS News Article. Available online: https://www.nocutnews.co.kr/news/4846285 (accessed on 1 April 2021).
2. Szabo, N. Smart Contracts: Building for Digital Markets. 1996. Available online: http://www.truevaluemetrics.org/DBpdfs/BlockChain/Nick-Szabo-Smart-Contracts-Building-Blocks-for-Digital-Markets-1996-14591.pdf (accessed on 30 April 2021).
3. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 30 April 2021).
4. Buterin, V. *A Next Generation Smart Contract & Decentralized Application Platform*; Ethereum Foundation: Bern, Switzerland, 2014.
5. Androulaki, E.; Cachin, C.; Ferris, C.; Muralidharan, S.; Murthy, C.; Nguyen, B.; Sthi, M.; Stathakopoulou, C. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In Proceedings of the Thirteenth EuroSys Conference (EuroSys '18), Porto, Portugal, 23–26 April 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 1–15. [CrossRef]
6. Rajput, A.R.; Li, Q.; Ahvanooey, M.T.; Masood, I. EACMS: Emergency Access Control Management System for Personal Health Record Based on Blockchain. *IEEE Access* **2019**, *7*, 84304–84317. [CrossRef]
7. Liu, Q.; Guan, Q.; Yang, X.; Zhu, H.; Green, G.; Yin, S. Education-Industry Cooperative System Based on Blockchain. In Proceedings of the 2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN), Shenzhen, China, 15–17 August 2018; pp. 207–211. [CrossRef]
8. Mohite, A.; Acharya, A. Blockchain for government fund tracking using Hyperledger. In Proceedings of the 2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS), Belgaum, India, 21–22 December 2018; pp. 231–234. [CrossRef]
9. Jiang, S.; Cao, J.; Wu, H.; Yang, Y.; Ma, M.; He, J. BlocHIE: A BLOCkchain-Based Platform for Healthcare Information Exchange. In Proceedings of the 2018 IEEE International Conference on Smart Computing (SMARTCOMP), Taormina, Sicily, Italy, 18–20 June 2018; pp. 49–56. [CrossRef]
10. Jiang, S.; Cao, J.; Wu, H.; Yang, Y. Fairness-based Packing of Industrial IoT Data in Permissioned Blockchains. *IEEE Trans. Ind. Inform.* **2020**, 1. [CrossRef]
11. Onik, M.H.; Miraz, M.H.; Kim, C. A Recruitment and Human Resource Management Technique Using Blockchain Technology for Industry 4.0. In Proceedings of the Smart Cities Symposium, SCS-2018, Manama, Bahrain, 22–23 April 2018. [CrossRef]
12. Pinna, A.; Ibba, S. A blockchain-based Decentralized System for proper handling of temporary Employment contracts. *ArXiv* **2017**, arXiv:abs/1711.09758.
13. Lallai, G.; Pinna, A.; Marchesi, M.; Tonelli, R. Software Engineering for DApp Smart Contracts managing workers Contracts. In Proceedings of the 3rd Distributed Ledger Technology Workshop, DLT 2020, Ancona, Italy, 4 February 2020.
14. Kosba, A.; Miller, A.; Shi, E.; Wen, Z.; Papamanthou, C. Hawk: The blockchain model of cryptography and privacy-preserving smart contract. In Proceedings of the 2016 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–26 May 2016; pp. 839–858. [CrossRef]
15. Song, D.X.; Wagner, D.; Perrig, A. Practical techniques for searches on encrypted data. In Proceedings of the 2000 IEEE Symposium on Security and Privacy, S&P 2000, Berkeley, CA, USA, 14–17 May 2000; pp. 44–55. [CrossRef]
16. Boneh, D.; Di Crescenzo, G.; Ostrovsky, R.; Persiano, G. Public Key Encryption with Keyword Search. In *Advances in Cryptology—EUROCRYPT 2004*; EUROCRYPT 2004, Lecture Notes in Computer Science; Cachin, C., Camenisch, J.L., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; Volume 3027. [CrossRef]
17. Boneh, D.; Waters, B. Conjunctive, Subset, and Range Queries on Encrypted Data. In *Theory of Cryptography*; TCC 2007. Lecture Notes in Computer Science; Vadhan, S.P., Ed.; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4392. [CrossRef]
18. Boneh, D.; Franklin, M. Identity-based Encryption from the Weil Pairing. *SIAM J. Comput.* **2003**, *32*, 586–615, Extended abstract in Crypto 2001. [CrossRef]
19. Goh, E.-J. Secure Indexes. Cryptology ePrint Archive. Report 2003/216. 2003. Available online: https://eprint.iacr.org/2003/216.pdf (accessed on 30 April 2021).
20. Liu, C.; Zhu, L.; Chen, J. Efficient searchable symmetric encryption for storing multiple source data on cloud. *J. Netw. Comput. Appl.* **2017**, *86*, 3–14. [CrossRef]
21. Wang, C.; Cao, N.; Ren, K.; Lou, W. Enabling secure and efficient ranked keyword search over outsourced cloud data. *IEEE Trans. Parallel Distrib. Syst.* **2012**, *23*, 1467–1479. [CrossRef]
22. Wang, C.; Cao, N.; Li, J.; Ren, K.; Lou, W. Secure ranked keyword search over encrypted cloud data. In Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems, Genoa, Italy, 21–25 June 2010; pp. 253–262. [CrossRef]
23. Cao, N.; Wang, C.; Li, M.; Ren, K.; Lou, W. Privacy-preserving multikeyword ranked search over encrypted cloud data. *Ieee Trans. Parallel Distrib. Syst.* **2014**, *25*, 222–233. [CrossRef]
24. Rajan, D.P.; Alexis, S.J.; Gunasekaran, S. Dynamic multi-keyword based search algorithm using modified based fully homomorphic encryption and Prim's algorithm. *Clust. Comput.* **2019**, *22*, 11411–11424. [CrossRef]

25. Yin, F.; Zheng, Y.; Lu, R.; Tang, X. Achieving Efficient and Privacy-Preserving Multi-Keyword Conjunctive Query Over Cloud. *IEEE Access* **2019**, *7*, 165862–165872. [CrossRef]
26. Wu, D.N.; Gan, Q.Q.; Wang, X.M. Verifiable public key encryption with keyword search based on homomorphic encryption in multi-user setting. *IEEE Access* **2018**, *6*, 42445–42453. [CrossRef]
27. Cao, Q.; Li, Y.; Wu, Z.; Miao, Y.; Liu, J. Privacy-preserving conjunctive keyword search on encrypted data with enhanced fine-grained access control. *World Wide Web* **2020**, *23*, 959–989. [CrossRef]
28. Hu, S.; Cai, C.; Wang, Q.; Wang, C.; Luo, X.; Ren, K. Searching an Encrypted Cloud Meets Blockchain: A Decentralized, Reliable and Fair Realization. In Proceedings of the IEEE INFOCOM 2018—IEEE Conference on Computer Communications, Honolulu, HI, USA, 15–19 April 2018; pp. 792–800. [CrossRef]
29. Jiang, S.; Cao, J.; McCann, J.A.; Yang, Y.; Liu, Y.; Wang, X.; Deng, Y. Privacy-Preserving and Efficient Multi-Keyword Search over Encrypted Data on Blockchain. In Proceedings of the 2019 IEEE International Conference on Blockchain (Blockchain), Atlanta, GA, USA, 14–17 July 2019; pp. 405–410. [CrossRef]
30. Gentry, C. A fully homomorphic encryption scheme. Ph.D. Thesis, Stanford University, Stanford, CA, USA, 2009. Available online: https://crypto.stanford.edu/craig/craig-thesis.pdf (accessed on 30 April 2021).
31. Gentry, C. Fully homomorphic encryption using ideal lattices. In Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, 31 May–2 June 2009; Association for Computing Machinery: New York, NY, USA, 2009; pp. 169–178. [CrossRef]
32. van Dijk, M.; Gentry, C.; Halevi, S.; Vaikuntanathan, V. Fully Homomorphic Encryption over the Integers. In *Advances in Cryptology—EUROCRYPT 2010*; EUROCRYPT 2010. Lecture Notes in Computer Science; Gilbert, H., Ed.; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6110. [CrossRef]
33. Koblitz, N. Elliptic Curve Cryptoystems. *Math. Comput.* **1987**, *48*, 203–209. [CrossRef]