Grant Agreement No.: 952697
Call: H2020-SU-ICT-2018-2020
Topic: SU-ICT-02-2020
Type of action: RIA

D4.3: ASSURED BLOCKCHAIN-BASED CONTROL SERVICES AND CRYPTO FUNCTIONS FOR DECENTRALIZED DTA STORAGE, SHARING AND ACCESS CONTROL

Revision: v.1.0

Work package	WP 4
Task	Task 4.2 – 4.3
Due date	28/02/2022
Deliverable lead	TUDE
Version	1.0
Authors	Kaitai Liang (TUDE), Shihui Fu (TUDE)
Reviewers	Sotiris Kousouris (S5) Liqun Chen (SURREY)
Abstract	In D4.3, we design and develop secure data search and sharing mechanism via the use of advanced lightweight cryptographic primitives for enabling decentralized data integrity, access control and multi-level data sharing between different stakeholders in the context of dynamic supply chains. This will allow users to directly search over encrypted data stored on public ledger by using a Dynamic SSE scheme, while the data search and sharing over the private ledger can be also controlled and monitored.
Keywords	Search, Data Sharing and Data Access

Document Revision History

Version	Date	Description of change	List of contributors
v0.1	15.12.2021	ToC	Shihui Fu (TUDE)
v0.2	07.01.2022	State-of-the-art analysis of Searchable Encryption Schemes and their possible usage/integration in the context of ASSURED (Chapter 3)	Kaitai Liang, Shihui Fu (TUDE) Nada El Kassem (SURREY) Stefanos Venios (S5)
v0.3	14.01.2022	Mapping of the ASSURED secure data sharing and management requirements with possible SE techniques so as to identify the best possible approach to follow (Chapter 2)	Kaitai Liang, Shihui Fu (TUDE) Stefanos Venios (S5) Thanassis Giannetsos (UBITECH)
v0.4	21.01.2022	Description of the first draft of the Dynamic Searchable Encryption scheme to be employed in ASSURED (Chapter 4)	Kaitai Liang, Shihui Fu (TUDE) Nada El Kassem (SURREY)
v0.5	04.02.2022	Design of the ASSURED Data Storage Engine so as to be able to handle the confidentiality and integrity requirements, due to the sensitivity of the attestation-related data stored, as well as the storage requirements of the designed SE scheme (Chapter 5)	Sotiris Kousouris, Stefanos Venios (S5) Thanassis Giannetsos (UBITECH)
v0.6	11.02.2022	Update and finalization of the ASSURED SE scheme (Chapter 4)	Kaitai Liang, Shihui Fu (TUDE)
v0.7	18.02.2022	Description of the necessary cryptographic primitives needed as building blocks for ASSURED SE and secure data storage (Chapters 2 and 3)	Kaitai Liang, Shihui Fu (TUDE) Nada El Kassem (SURREY) Stefanos Venios (S5)
v0.9	24.02.2022	Review the document	Sotiris Kousouris (S5) Liqun Chen (SURREY)
v1.0	27.02.2022	Finalisation of the document	Thanassis Giannetsos, Dimitris Karras (UBITECH)

Editors

Kaitai Liang (TUDE), Shihui Fu (TUDE)

Contributors (ordered according to beneficiary numbers)

Liqun Chen, Nada El Kassem (SURREY)

Kaitai Liang, Shihui Fu (TUDE)

Thanassis Giannetsos, Dimitris Papamartzivanos, George Misiakoulis (UBITECH)

Sotiris Koussouris, Stefanos Venios, Alexandros Tsaloukidis, Konstantinos Charalambous (SUITE5)

Funded by the Horizon 2020
Framework Programme of the European Union

DISCLAIMER

COPYRIGHT NOTICE

Project co-funded by the European Commission in the H2020 Programme		
Nature of the deliverable:	**R**	
Dissemination Level		
PU	Public, fully open, e.g. web	✓
CL	Classified, information as referred to in Commission Decision 2001/844/EC	
CO	Confidential to ASSURED project and Commission Services	

* R: Document, report (excluding the periodic and final reports)

DEM: Demonstrator, pilot, prototype, plan designs

DEC: Websites, patents filing, press & media actions, videos, etc.

OTHER: Software, technical diagram, etc.

Executive Summary

The main focus of this deliverable is on the decentralized data management and sharing over the ASSURED blockchain framework. Several main components and roles are considered in this deliverable, including the ASSURED Security Context Broker, data management over public and private data, and the data storage on the ASSURED data storage engine. Recall that from the deliverable D4.1 [12], data storage, management and sharing framework and the corresponding generic interactions were defined and introduced. The SCB acts as the adjunct point between external parties, blockchain ledgers (on-chain) and (off-chain) storage engine. Their connections and interactions are further elaborated in this deliverable.

We first review the secure data management in the ASSURED blockchain framework, including the roles, and types of data and actors, and the corresponding core security requirements. We further introduce the cryptographic technologies which are used in the design of secure data sharing in this deliverable. Specifically, we go through symmetric encryption, pseudorandom functions, Attribute-based Encryption (ABE), and Searchable Encryption (SE) components. In these components, we explain how they work and how they can merge with the ASSURED blockchain framework. We note that in the design of SE system we will leverage these components. We also explain the reasons of using these components.

After that, we proceed to the main contribution of this deliverable – the first stage design of the Dynamic Searchable Symmetric Encryption (DSSE) for the secure data search and sharing over ASSURED blockchain ledger. We introduce the DSSE system model and basic security guarantees during the data search and sharing. An overview of the design and the detailed descriptions are given. In those DSSE supported secure data operations, we first enable the SCB to receive attestation data bundle from the ASSURED blockchain peer(s). The SCB is allowed to use those attestation data (including encrypted raw data, attestation results, metadata, and related ID info) to interact with data storage engine in the sense that the encrypted raw data is stored on the engine and meanwhile, a linkable pointer is returned to the SCB. We note that the raw data is encrypted under the ABE format so as to provide the data confidentiality. The SCB can further calculate a hash value over the encrypted data, the pointer, and attestation ID. This hash value can be used to guarantee the integrity of the storage copy but also link the ID, pointer and attestation raw data together. Recall that the ASSURED blockchain framework can support two types of ledgers: public and private ledgers. For each of these types, we will design different approach of data storage and access. The DSSE scheme is mainly applied on the public ledger; while the data access control over the private ledger is based on ABAC. The cryptographic primitive, DSSE, can be securely used by the external stakeholders to perform search queries.

At last, the ASSURED data storage engine is defined. Some securing approaches on the data storage engine and storage options are further introduced.

The overall purpose of this deliverable is to provide a first stage development document on the use of SE, ABAC and data storage engine, and how these components interact together to provide security and privacy preservation on data search and data sharing. This will be used as input to the final stage design and implementation of the ASSURED decentralized data storage and management.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Scope and Purpose

This deliverable D4.3 “ASSURED Blockchain-based Control Services and Crypto Functions for Decentralized Data Storage, Sharing and Access Control” comes as the third deliverable of WP4 of the ASSURED project which aims to design and implement the blockchain-based cryptographic tools, APIs and functionalities to provide fully transparent and accountable secure information exchange. The deliverable corresponds to the Task 4.3 “Decentralized Data Storage, Sharing and Access Control Services”. The task is to define, design and implement the advanced lightweight crypto primitives to enable decentralized data integrity, access control and multi-level data sharing among stakeholders in the context of dynamic supply chains. From this perspective, we should mainly focus on the development of searchable encryption component. That is what this deliverable crucially reflects on. Recall that in the first deliverable of WP4, i.e. D4.1 [12], multiple data sharing and management components were defined, e.g., ASSURED blockchain infrastructure, the security context broker (SCB), attribute-based access control (ABAC), searchable encryption (SE), cloud-based backend, etc. And that deliverable described how these components should interact with each other in a general and generic approach. Starting from that, this deliverable first reviews those roles, actors, functionalities, types of data and the crucial security requirements in secure data management, which were proposed in D4.1 [12] and D1.4 [13]. From the reviewing, one may easily capture the understandings on the previous backgrounds. After that, the deliverable proceeds to the introduction of necessary cryptographic components, e.g., ABE and SE. Specifically, it introduces the basic definition, syntax, general model, security guarantees and the current state of the art in this research line. With the corresponding background knowledge, the deliverable presents the initial design of dynamic searchable symmetric encryption (DSSE) construction. A high-level and a detailed descriptions are given in the deliverable. We note that the use of DSSE is mainly targeting to the secure data search over ASSURED public ledger; as for the search on private ledger, we also provide an initial solution. At last, we provide the descriptions of the ASSURED data storage engine.

1.2 Relation to other WPs and Deliverables

D4.3 comes as the third deliverable of WP4 “Blockchain-based ASSURED Supply Chain Control Services and Trust Evidence Collection” and is acting as an important interaction adjunct for other deliverables in this work package. As illustrated in Figure 1.1, this deliverable first takes the input from the D4.1 [12]. D4.1 has defined ASSURED data management and sharing infrastructure, be-

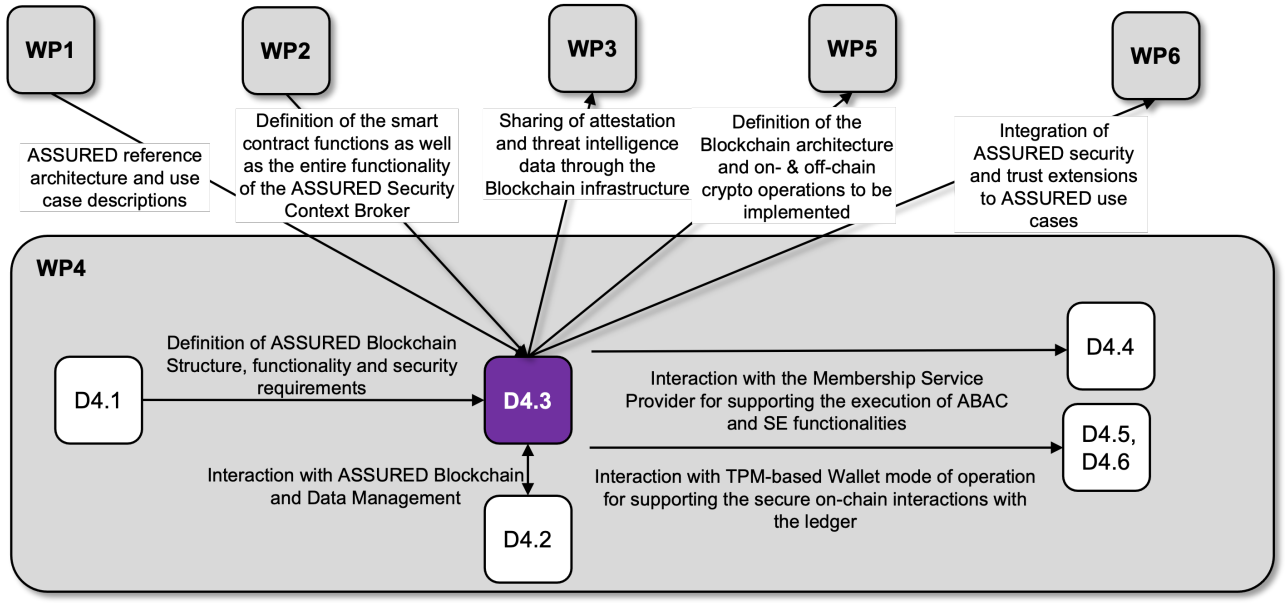

Figure 1.1: Relation of D4.3 to other WPs and Deliverables

haviours, and security requirements. Based on the guidelines output by the D4.1, this deliverable moves one step forward into the secure data sharing via the use of Searchable Encryption (SE) with the ASSURED blockchain framework and the descriptions of the data storage engine. D4.3 has strong connections with D4.2 [17] and D4.5 [18]. D4.2 introduces the main crypto primitives for data sharing: attribute-based encryption (ABE) and secure data management implementation diagrams. D4.5 explains how the interconnection is done within the blockchain framework using TPM wallet, and how the TPM wallet manages keys used by crypto primitives and blockchain users. D4.4 will be a further completed version of D4.3, in which we will consider more how all the TPM wallet, ABE, SE and ABAC interact as a whole service for different use cases. Further, D4.3 captures the data sharing behaviors and operations defined in the Task 1.5, and provides crypto primitives that will be implemented in the Task 5.2.

1.3 Deliverable Structure

The remaining of this deliverable is structured as follows. In **Chapter 2**, we review a high-level DLT data management components within the ASSURED framework, and the main requirements of DLT data storage, sharing and access control components. We explain the roles within the framework, trusted blockchain control services, trusted blockchain wallet, and how these elements interact with each other over different types of data. And further, we review all the security requirements of secure data management. In **Chapter 3** we present a thorough introduction on the cryptographic components used in our searchable encryption design. The state-of-the-art searchable encryption systems are introduced in this chapter. Later, in **Chapter 4**, we propose the DSSE scheme for the ASSURED secure data sharing. In this chapter, we introduce the core techniques and how the scheme works with different parties and components in the ASSURED blockchain framework. In **Chapter 5** we define the ASSURED data storage engine and how it provides the data storage and deployment options. Finally, **Chapter 6** concludes the deliverable.

Chapter 2

Secure Data Management in ASSURED

With the rapid development of cloud computing, cloud storage has enabled the provision of high data availability, easy access to data, and reduced infrastructure costs from outsourcing of data to services to relieve the burden of maintenance costs as well as the overhead of storing data locally. Moreover, users are able to access their data anywhere and at any time instead of having to use dedicated machines. In recent years, small and large business enterprises are moving their infrastructure to the cloud, because of two main reasons: massive storage capacity and accessibility. At the same time, the introduction of IoT and other similar infrastructures within organizations has led to the exponential generation of data, which fuels the movement of digitization of operations, which is based on exploiting all data that can be accessed to extract intelligence that can be used to improve operations. And it becomes evident that enterprise systems should capitalize not only on the internally produced data, but also on data they can access and acquire from external sources.

Cloud storage and IoT offer many advantages to users, however, there are still various security and privacy concerns. A remote server cannot be fully trusted because it may not only be curious about the users data but also abuse the data. Although users are able to access their data anywhere and at any time, when users outsource their data to a remote server, the physical access to the data is actually lost and the administration of the data is delegated to the server as well. Thus, it is necessary to guarantee the privacy of users sensitive data. The most common way of achieving privacy is to encrypt the data before outsourcing them. This approach provides end-to-end data privacy as soon as the data leave the users possession. While such a solution guarantees the privacy of sensitive data, it also brings difficulties for the server to perform any meaningful function, such as secure sharing, access control, search and some other data management functionalities, on the encrypted data.

For example, we consider a search function on encrypted data. A user sends query keywords to the server in order to retrieve corresponding documents. After searching, the server will return the search results to the user. However, during the search process, both the knowledge of the contents stored on the server and the query keywords are exposed to the semi-trusted server. Fortunately, encryption is a positive way to protect the privacy of users data, but at the same time it disrupts search functionality. A trivial way to search is to download all the ciphertexts, decrypt them, and then search on the plaintexts. However, this is impractical. Therefore, when we design the data sharing and management components of ASSURED, we needed a way to provide data confidentiality and preserve search functionality simultaneously.

Most of current secure and trusted systems focus on centralized approaches to provide secure data management. These systems strongly relies on a or some trusted parties to handle data sharing and meanwhile, they are easily vulnerable to many types of attacks [35]. This motivates

us to use a series of new technologies to achieve the goal of secure data management in ASSURED.

In ASSURED, we provide a secure, trusted, auditable and privacy-preserving platform for operational and security-related data management including their secure storage, access control and searching. To this end, we design smart contract based access control to verify data requester's attributes and related policies, use attribute-based access control (ABAC) to maintain ledger data access, attribute-based encryption (ABE) to guarantee final data copies' data confidentiality on the data storage engine, and dynamic searchable symmetric encryption (DSSE) as a privacy-preserving data searching service.

In this chapter, we are going to review the roles, DLT data management and security requirements for the ASSURED framework which have been defined in the Deliverable 4.1 [13], setting the tone for the consequent chapters that elaborate the design of our secure data sharing and storage components.

2.1 Roles of Secure Data Management in ASSURED

ASSURED aims to protect data generated within consortium and on supply chain against leakage or tamper, whilst at the same time provide data availability to the internal and external parties from the data value chains. Internal and external storage via ledger infrastructures and cloud-based backend (called **ASSURED Data Storage Engine** in the ASSURED framework, we will cover the data storage engine in more details in Chapter 5) can be used to maintain data confidentiality and sharing, track and audit data correctness and processing with specified security and privacy policies. In this way data and their sources are maintained to be inline with the specified privacy, applications and services.

As has been mentioned in the Deliverables 1.1 [15] and 1.2 [14], using the Blockchain Distributed Ledgers to guarantee data sharing and security policy enforcement, is considered one of the main contributions delivered by the ASSURED project. ASSURED attempts to enhance data privacy, storage, sharing and access control mechanisms via the combination of cryptogrpahic primitives and blockchain techniques. The DLT platform and off-chain data storage engine mainly offer secure data storage and management services. In terms of secure data sharing, we will mainly make use of trusted computing components, i.e. TPM, ABAC, ABE and searchable encryption technologies to ensure that all the data related operations are auditable and compliant with security and privacy policies.

As has been described in Deliverables 4.1 [12]. ASSURED will leverage private and public ledgers to handle on-chain data management. The former, controlling internal data sharing, will storage smart contracts between the ASSURED Platform and the internal components, based on the details of the data sharing, and further, some related data including attestation results, attestation ID, will be put on the private ledger. The latter, i.e. the public ledger, is mainly used for handling data sharing to external parties. It is used to record some encrypted form of metadata and pointers which can help external parties to have fuzzy search over the attestation raw data. In addition to the on-chain data management, ASSURED will also make use of the off-chain data storage via the use of cloud-based data storage engine. That data storage engine will be used to store the encrypted attestation raw data under the ABE. Figure 2.1 shows the general framework of on-chain and off-chain data sharing.

Below, we are going to review the main components or services related to data management as shown in Figure 2.1. We first use the hardware trust anchors (TPM) in supporting secure

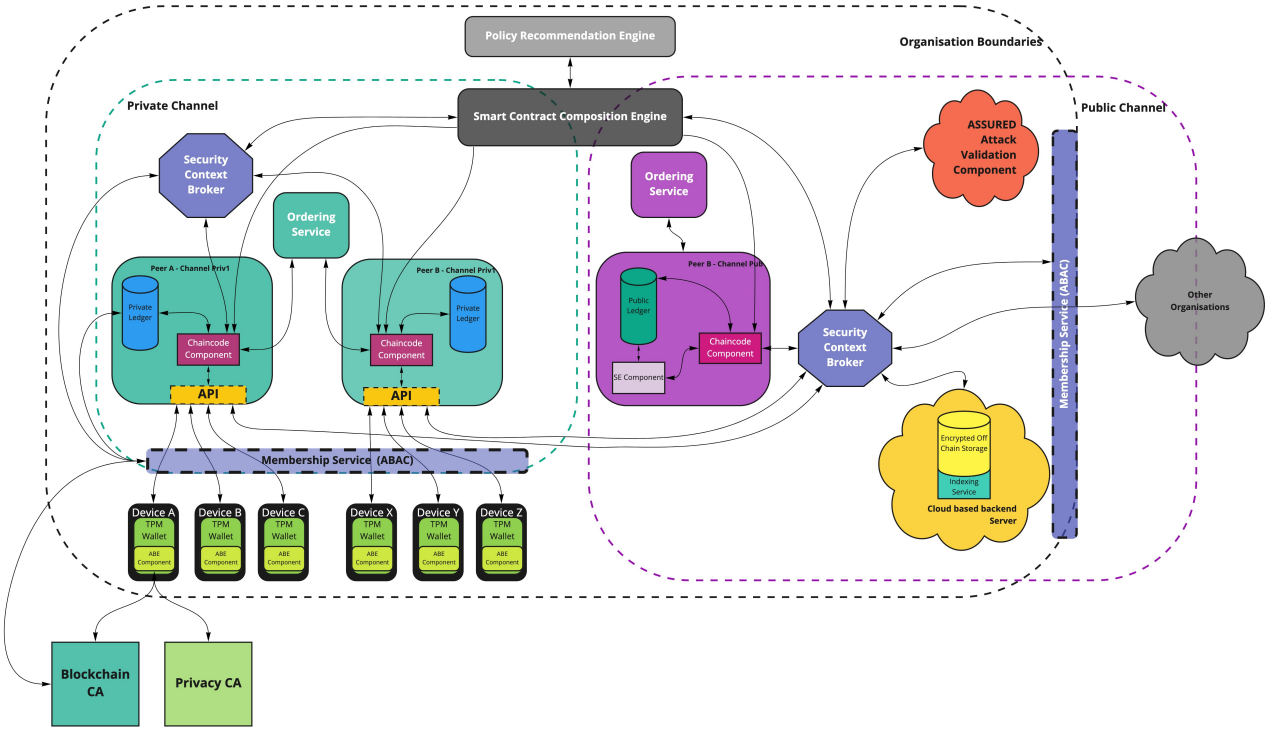

Figure 2.1: ASSURED DLT Architecture (from D4.1 [12])

information storage and authentication.

TPM Embeded in a Device: TPMs are used as a useful building block for ASSURED data sharing mechanisms and they enhance the trust on the data management operations but also strengthen the secure use of cryptographic primitives. The novel integration of the TPM technology with ASSURED blockchain and crypto functions can provide the state-of-the-art of blockchain-based data sharing services:

- **Connecting Cryptographic Primitives:** as for the on-chain and off-chain data sharing, we mainly leverage symmetric encryption, ABE and searchable encryption. These components will be further introduced in Chapter 3.
- **Secure Information Storage:** a user can store private/secret keys into a TPM, and, when authorized by the user, the TPM allows access to the user's secrets.

Trusted TPM-based Blockchain Wallet: In the ASSURED framework, TPMs are also the basis for trusted blockchain wallets:

- **Authentication:** it can provide strong user authentication and to securely store blockchain users' (e.g., devices, external parties) credentials based on the TPM's secure key storage.
- **Access and Key Control:** it can control and authorize access to private or public ledger channels based on the authentication process. Specifically, ASSURED leverages the TPM-based wallet for each entity to facilitate a decentralized ABE and searchable encryption mechanisms where the necessary keys are managed locally.

Secure Data Search: This service mainly includes the interactions with the security context broker (SCB), the ASSURED public ledger, ASSURED data storage engine and the searchable encryption component. As information stored in the public Ledger will be encrypted, it is essential

to provide a service that allows interested stakeholders to be able to locate whether information of their interest exists within an organization and then connect them to the right retrieval points which are given by the data storage engine and stored in the encrypted form on the public ledger. In order to facilitate this need, within each public ledger constructed within the ASSURED blockchain network, we will deploy a dynamic searchable symmetric encryption component that will allow interested stakeholders to perform queries on top of encrypted metadata that are stored in the public ledger and accompany the different attestation information generated by the devices. These metadata will be provided by the SCB in an encrypted manner, and this component will allow stakeholders to request a special search token corresponding to a or some keywords to perform queries over the encrypted metadata. In case the token of the querying is valid, and the requested information exists on the ledger, this will be revealed to the query party, reverting them to the storage engine where they should query to get the full attestation report.

2.2 Types of Data and Actors

Types of Data. We briefly review the types of the data which are mainly involved into the secure data sharing and storage stages in ASSURED. As described in D1.4 [13, Table 1], we focus on the following data in this deliverable:

- Attestation Raw Data: This is the data collected from the deployed devices and assets (sources) and can include the information related to attestation.
- Attestation Result Data: This is the result data of analyzing from attestation, and this will only be used to indicate a result of the attestation, e.g., a failed attestation.
- Identity Data: This is a type of data we use to set identify attestation, which we can also call it attestation ID (identifier).

Data Categories. In ASSURED we have two main data categories: one is stored on ledger which we call **on-chain data**, and the other is the **off-chain data** which is mainly stored on the ASSURED data storage engine. As for the on-chain data, we further separate two different cases: the data on the private ledger, and those on the public ledger. The data stored on the private ledger are accessible for the private channel users. For convenience, we store the attestation results, unencrypted pointers, and other related information, e.g., attestation ID and hash values, on the private ledger, where the pointers are those references used to retrieve encrypted attestation raw data from the data storage engine. We also store data on the public ledger, so that external parties can perform some level (fuzzy) of secure search. Those data include encrypted index structure, encrypted pointers and related information, e.g., hash values. We note that the detailed use of the data will be seen in Chapter 4 while introducing the DSSE scheme. As for the off-chain data, they are mainly stored on the data storage engine, in ABE encrypted format. These data are usually referred to as the (encrypted) attestation raw data. We further will elaborate how the on-chain and off-chain data interact between ledgers and data storage engine.

Data Format. Recall that in D1.4 [13] we define three major groups of data format:

- Unstructured Data: This refers to any dataset without a reliable structure from which we can extract other data of our interest.
- Structured Data: This is the data that has fixed and well-known format and structure that allows relationships to be established.

- Semi-structured Data: This is the data that has a fixed format but with a non-strict organization, such as XML and JSON.

In this deliverable, we mainly focus on the last two types of data format. As for the structured data, we refer to those encrypted data under well-formed and well-studied cryptographic mechanisms, e.g., ABE, searchable encryption, hash function. Besides, we will need a keyword dictionary giving to external parties so that they are able to know which keywords they may use for search queries. This dictionary is regarded as structured data. We will also use the semi-structured data which includes attestation raw data.

Data Actors. We will use some significant actors related to the data sharing in ASSURED.

- Data Subject: This role reflects the identified asset who is providing operational and security raw data for further processing and is the actual data sources. Here we mainly consider devices as this actor.
- Data Controller: The person or (software) asset which alone or jointly with other can determine the purpose and means of the processing of the collected operational raw data. We here consider the ASSURED blockchain administrator, Security Context Broker (SCB), and data storage engine, in which SCB mainly participates into data sharing and searching.
- Data Processor: A natural person, public authority, agency or other body party that processes operational and/or security related data. For this role, we consider the SCB, ASSURED blockchain peers/orderers, and the data storage engine administrator.
- Data Recipient: A natural person, public authority, agency or another body (internal or external to current network), to which the operational and security related data are disclosed, whether a third party or not. In this case, this role can be naturally taken by devices, external parties, the SCB and data storage engine.
- Third Party: A natural person, public authority, agency or body (external to the supply chain) other than the data subject, controller, processor and persons who are authorized to process personal data. For example, the interested stakeholders who want to search interested data and further issue related data queries.

2.3 Security Requirement of Secure Data Management

We review the security requirements for the ASSURED secure and accountable data management, in Table 2.1, through the use of DLT technologies, Attribute-Based Encryption/Access Control and Searchable Encryption components. As aforementioned in D1.4 [13] and D4.1 [12], we should guarantee the data confidentiality, integrity, searchability and multi-level access control. We note that here we mainly focus on those requirements that are related to data storage and sharing.

Table 2.1: Security Requirements of Secure Data Management

Components	Descriptions
DLT Infrastructure	DLT-SCT-03 – The DLT infrastructure shall provide the necessary availability guarantees for the data by utilising an off-chain data storage facility

Table 2.1: Security Requirements of Secure Data Management (Continued)

DLT Infrastructure	DLT-SCT-05 – The original and full copy of a given encrypted data shall be stored on the off-chain storage to allow the ledgers to be performant and meet availability SLAs
	DLT-SCT-07 – The encrypted data stored on the ledgers shall be a pointer to the actual data in the off-chain data storage engine
	DLT-SCT-30 – The data to be stored stored on the ASSURED DLT Infrastructure shall be protected by strong cryptographic primitives coming out of the TPM of the devices that also perform the attestation
	DLT-SCT-50 – The attestation data stored on the ledger shall not be able to be removed or amended once the initial transaction has been executed
	DLT-SCT-52 – The data sharing event and status shall be recorded on the ledger for later auditing purposes
	DLT-SCT-54 – The attestation data stored on the ledger shall not be altered by any action/user
	DLT-SCT-55 – The data stored on the ledger shall be validated by a set of peers during the transaction
	DTL-DEC-06 – Data stored on the ledgers shall enable confidentiality by being encrypted
Security Context Broker	DLT-SCT-08 – The off-chain data storage shall be accessible only via the SCB which acts as a proxy between the off-chain data storage engine and the requester
	DLT-SCT-13 -– The SCB shall securely obtain the storage pointer of an attestation report and pass it to the private network.
	DLT-SCT-14 – The SCB shall securely obtain the storage pointer of an attestation report and pass it to the private network/Searchable Encryption component
	DLT-SCT-17 – The SCB shall produce the metadata for the attestation assets to be encrypted by the Searchable Encryption Component
Attribute-Based Access Control	DLT-SCT-21 – The ABAC layer shall be executed as part of the functionality to be offered by the SCB
	DLT-SCT-22 – Data stored in the ASSURED DLT Infrastructure can only be read by entities that possess the appropriate attributes
	DLT-SCT-24 – The ABAC shall verify via the credentials of an entity that have been obtained by the Blockchain CA and the policies defined by policy engine
	DLT-SCT-25 – The ABAC shall provide the different permissions to the parties that want to access the DLT infrastructure
Attribute-Based Encryption	DLT-SHA-14 – The data coming out of the devices shall be encrypted using ABE

Table 2.1: Security Requirements of Secure Data Management (Continued)

Attribute-Based Encryption	DLT-SCT-34 – The encryption methods to be utilized by the TPMs shall utilize an ABE scheme to allow data to be read (decrypted) by entities that possess the correct attributes
	DLT-SCT-35 – Decryption of data fetched from the DLT Infrastructure shall happen at the device level of the data requester using its attributes and keys stored in the TPM
	DLT-SCT-40/DLT-SCT-41 – The ABE component shall enable devices to perform the corresponding encryption/decryption with the help of TPM
	DLT-SCT-42 – The encryption performed shall be based on attribute and policy belonging to the TPM devices
	DLT-SCT-43 – The decryption key shall be protected and issued by the TPM based on attributes provided by the device
	DLT-SCT-44 – The TPM shall use the main secret key to generate an attribute-based key that helps devices to perform valid decryption
Searchable Encryption	DLT-SCT-36 – The encrypted data to be stored in the ASSURED DLT Infrastructure shall be made searchable in its encrypted state, using SE functionality
	DLT-SCT-38 – The searchable encryption functionality shall be provided at the public peer level
	DLT-SCT-39 – Queries over the searchable data shall be performed via the SCB acting as a proxy between the requester and the searchable encryption component
	DLT-SCT-45 – The searchable encryption functionality shall generate a keyword index structure and further encrypt the structure along with the data
	DLT-SCT-46 – The secret key of searchable encryption component shall be hosted and protected by the SCB's TPM
	DLT-SCT-47 – The searchable encryption component shall return replies to queries performed by users via the SCB

Chapter 3

Cryptographic Technologies

In this chapter, we elaborate the fundamental cryptographic primitives that are used in searchable encryption service in ASSURED. These functions will be used to provide secure data access and could be enhanced through the utilization TPM (see the deliverable D4.5 [18]). Given these functions, ASSURED will offer a set to cryptographic abstractions, i.e., advanced crypto-based operations supporting the blockchain-based data searching and sharing.

3.1 Symmetric Encryption

In the context of symmetric encryption, i.e., private-key encryption, two parties share a key and use that key when they want to communicate secretly. One party can send a message, or plaintext, to the other by using the shared key to encrypt the message and thus obtain a ciphertext that is transmitted to the receiver. The receiver uses the same key to decrypt the ciphertext and recover the original message. The same key is used to convert the plaintext into a ciphertext and back and both parties hold the same key that is used for encryption and decryption. This is in contrast to asymmetric, or public-key encryption (introduced in D4.2 [17, Chapter 2]), where encryption and decryption use different keys. We will discuss this in more detail here as our searchable encryption scheme relies heavily on symmetric encryption.

Formally, a private-key encryption scheme is defined by specifying a message space $\mathcal{M}$ along with three algorithms: a procedure for generating keys (Gen), a procedure for encrypting (Enc), and a procedure for decrypting (Dec). The message space $\mathcal{M}$ defines the set of "legal" messages, i.e., those supported by the scheme. The algorithms of the scheme have the following functionality:

1. The key-generation algorithm Gen is a probabilistic algorithm that outputs a key k chosen according to some distribution.
2. The encryption algorithm Enc takes as input a key k and a message m and outputs a ciphertext c. We denote by $\mathsf{Enc}_k(m)$ the encryption of the plaintext m using the key k.
3. The decryption algorithm Dec takes as input a key k and a ciphertext c and outputs a plaintext m. We denote the decryption of the ciphertext c using the key k by $\mathsf{Dec}_k(c)$.

A symmetric encryption scheme must satisfy the following correctness requirement: for every key k output by Gen and every message $m \in \mathcal{M}$, it holds that

$$\mathsf{Dec}_k(\mathsf{Enc}_k(m)) = m.$$

The set of all possible keys output by the key-generation algorithm is called the key space and is denoted by $\mathcal{K}$. Almost always, Gen simply chooses a key uniformly from the key space.

An encryption scheme should be designed to be secure even if an eavesdropper knows all the details of the scheme, so long as the attacker doesn't know the key being used. Stated differently, security should not rely on the encryption scheme being secret; instead, Kerckhoffs' principle demands that security rely solely on secrecy of the key.

CPA-Security. Chosen-plaintext attacks (CPA) capture the ability of an adversary to exercise (partial) control over what the honest parties encrypt. Imagine a scenario in which two honest parties share a key k, and the attacker can influence those parties to encrypt messages $m_1, m_2, \ldots$ and send the resulting ciphertexts over a channel that the attacker can observe. At some later point in time, the attacker observes a ciphertext corresponding to some unknown message m encrypted using the same key k; let us even assume that the attacker knows that m is one of two possibilities m_0, m_1. Security against chosen-plaintext attacks means that even in this case the attacker cannot tell which of those two messages was encrypted with probability significantly better than random guessing. In practical applications, if AES is a secure pseudorandom permutation (we cannot prove it at the current stage, the existence of pseudorandom permutations depends on the existence of one-way functions), then AES in Cipher Block Chaining (CBC) mode and in Counter (CTR) mode are CPA-secure.

Even though chosen-plaintext attacks allow an adversary to control what gets encrypted, the adversary in that setting is still limited to passively observing ciphertexts transmitted by the honest parties. We have a stronger security definition which makes any modification of an encryption c "useless", called CCA-security.

CCA-Security. In a chosen-ciphertext attack (CCA) game, we provide the adversary not only with the ability to encrypt messages of its choice (as in a chosen-plaintext attack), but also with the ability to decrypt ciphertexts of its choice (except the challenge ciphertext c that is received from the challenger). Formally, we give the adversary access to a decryption oracle in addition to an encryption oracle. CCA-secure encryption scheme can be constructed from a CPA secure encryption and a secure message authentication code (MAC) by following the encrypt-then-authenticate paradigm. If AES is a secure pseudorandom permutation, then AES in Galois/Counter Mode (GCM) is CCA-secure.

Anonymous. A symmetric encryption scheme is anonymous if given two ciphertexts, one cannot determine whether they were encrypted under the same key.

Jumping ahead, when deploying the searchable encryption service, the SCB will be responsible for generating the index-based database related to the search function and the encrypted documents (i.e., the location (pseudo-) pointers, see Chapters 4 and 5), then send them to the public ledger. The documents can be encrypted using an independent symmetric encryption scheme. As with the DSSE scheme described in Chapter 4, we actually make use of the symmetric encryption scheme as a black-box and do not rely on its concrete construction. In addition, when uploading/outsourcing the attestation raw data, the devices can use any symmetric encryption scheme to encrypt the attestation raw data, and then further use an ABE scheme to encrypt the private key (of the symmetric encryption scheme). Likewise, we make use of the symmetric encryption scheme also as a black-box. The only security requirement for the symmetric encryption scheme being used in our searchable symmetric encryption scheme is that it should be anonymous and CPA-secure (see Section 4.3.2).

However, for efficiency, we recommend using the well-known 256-bit AES in CBC mode as the

symmetric encryption algorithm/scheme we are going to use in our searchable symmetric encryption scheme because AES-CBC is anonymous and CPA-secure.

3.2 (Pseudo-) Random Functions

We denote by $U(\mathcal{D}, \mathcal{R})$ the uniform distribution over all functions RO : $\mathcal{D} \to \mathcal{R}$ where $\mathcal{R}$ is finite (implicitly defined by the probabilistic algorithm that assigns, uniformly and independently at random, an $\ell_{\mathcal{R}}(\lambda)$-bit string to each new input). If RO is sampled from $U(\mathcal{D}, \mathcal{R})$, then we say that RO is a random oracle.

A keyed function $F : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ is a two-input function, where $\mathcal{K}$ and $\mathcal{R}$ are finite, and the first input is called the key and typically denoted by K. $\mathcal{K}$ is the key space of F, $\mathcal{D}$ is its domain, and $\mathcal{R}$ is the range of F. They respectively have size $|K| = 2^{\ell_{\mathcal{K}}(\lambda)}$, $|D| = 2^{\ell_{\mathcal{D}}(\lambda)}$, and $|R| = 2^{\ell_{\mathcal{R}}(\lambda)}$, with $\ell_{\mathcal{K}}, \ell_{\mathcal{D}}, \ell_{\mathcal{R}} : \mathbb{N} \to \mathbb{N}$. For $K \in \mathcal{K}$, we denote F_K the function that is the partial evaluation of F on K, namely

$$\begin{aligned} F_K : \mathcal{D} &\to \mathcal{R} \\ x &\mapsto F(K, x). \end{aligned}$$

We say F is efficient if there is a polynomial-time algorithm that computes $F(K, x)$ given K and x.

Definition 1 (Pseudorandom Function)**.** *An efficient, keyed function $F : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ is a pseudorandom function (PRF) if for all probabilistic polynomial-time distinguishers $\mathcal{A}$, there is a negligible function* negl *such that:*

$$\mathsf{Adv}^{\mathrm{prf}}_{\mathcal{A},F}(\lambda) = \left| \Pr\left[K \leftarrow\!\!\$\, \mathcal{K} : \mathcal{A}^{F_K(\cdot)}(1^\lambda) = 1\right] - \Pr\left[f \leftarrow\!\!\$\, U(\mathcal{D}, \mathcal{R}) : \mathcal{A}^{f(\cdot)}(1^\lambda) = 1\right] \right| \leq \mathsf{negl}(\lambda),$$

where the probability is also taken over the randomness of $\mathcal{A}$.

The dynamic SSE described in Chapter 4 will employ two random oracles to mask the (search, addition and deletion) tokens and the information related to the document identifiers (i.e., the location (pseudo-) pointers, see Chapters 4 and 5). Concretely, when building the index-based database related to the search function, the SCB will use the random oracles to generate random strings related to the keyword w, and then mask the identifiers and the storage location information of documents containing the keyword w. When adding/deleting a document, the SCB also uses the random oracles to generate random strings to mask the corresponding addition/deletion tokens. So, basically, we use random oracles to mask relevant information to avoid unnecessary information leakage and achieve a high level of security and privacy.

The random oracles can be implemented using HMAC-SHA256 (the HMAC construction is first described by Bellare, Canetti and Krawczyk [2]). The first parameter passed to the random oracle is used as a key to the HMAC, and the second parameter is used as input to the HMAC.

3.3 Attribute-Based Encryption

Attribute-based encryption (ABE) is a relatively recent approach that reconsiders the concept of public-key cryptography. In traditional public-key cryptography, a message is encrypted for a

specific receiver using the receiver's public-key. Identity-based cryptography and in particular identity-based encryption (IBE) changed the traditional understanding of public-key cryptography by allowing the public-key to be an arbitrary string, e.g., the email address of the receiver. ABE goes one step further and defines the identity not atomic but as a set of attributes, e.g., roles, and messages can be encrypted with respect to subsets of attributes (key-policy ABE) or policies defined over a set of attributes (ciphertext-policy ABE). The key issue is that someone should only be able to decrypt a ciphertext if the person holds a key for "matching attributes" where user keys are always issued by some trusted party.

Ciphertext-Policy ABE. In ciphertext-policy attribute-based encryption (CP-ABE) a user's private-key is associated with a set of attributes and a ciphertext specifies an access policy over a defined universe of attributes within the system. A user will be ale to decrypt a ciphertext, if and only if his attributes satisfy the policy of the respective ciphertext. Policies may be defined over attributes using conjunctions, disjunctions and (k, n)-threshold gates, i.e., k out of n attributes have to be present (there may also be non-monotone access policies with additional negations and meanwhile there are also constructions for policies defined as arbitrary circuits). For instance, let us assume that the universe of attributes is defined to be $\{A, B, C, D\}$ and user 1 receives a key to attributes $\{A, B\}$ and user 2 to attribute $\{D\}$. If a ciphertext is encrypted with respect to the policy $(A \wedge C) \vee D$, then user 2 will be able to decrypt, while user 1 will not be able to decrypt.

CP-ABE thus allows to realize implicit authorization, i.e., authorization is included into the encrypted data and only people who satisfy the associated policy can decrypt data. Another nice features is, that users can obtain their private keys after data has been encrypted with respect to policies. So data can be encrypted without knowledge of the actual set of users that will be able to decrypt, but only specifying the policy which allows to decrypt. Any future users that will be given a key with respect to attributes such that the policy can be satisfied will then be able to decrypt the data.

Key-Policy ABE. Key-policy attribute-based encryption (KP-ABE) is the dual to CP-ABE in the sense that an access policy is encoded into the users secret key, e.g., $(A \wedge C) \vee D$, and a ciphertext is computed with respect to a set of attributes, e.g., $\{A, B\}$. In this example the user would not be able to decrypt the ciphertext but would for instance be able to decrypt a ciphertext with respect to $\{A, C\}$. An important property which has to be achieved by both, CP- and KP-ABE is called collusion resistance. This basically means that it should not be possible for distinct users to "pool" their secret keys such that they could together decrypt a ciphertext that neither of them could decrypt on their own (which is achieved by independently randomizing users' secret keys).

3.3.1 ABE in ASSURED Framework

In the ASSURED framework, a TPM is used to support a KP-ABE where the secret key of a blockchain user and the ciphertext are dependent on the user attributes. In such a system, the decryption of a ciphertext is possible only if the set of attributes of the user key matches the attributes of the ciphertext. The blockchain user (being authenticated to a ledger) can only have decryption rights to the encrypted data stored on the ledger only if the user possesses some attributes that make correct decryption. This allows data owners to share data safely with the designated group of users rather than a third party or other users.

For instance, data encryption and access control is a topmost requirement in the "Public Safety" ASSURED use case and needs to be complemented by specified policies and access control

mechanisms to guarantee that only valid data retrievers can decrypt them. To achieve this requirement, an assigned administrator pre-defines a policy access list that will be merged on the smart contract to auto-check if different domains'/areas' operational level's entities could have access to the operational data. Specified policies and access control mechanisms would be implemented through ABE schemes to guarantee that only eligible entities can retrieve data from the cloud if needed.

For the "Smart Manufacturing" ASSURED use case, data is stored on IoT gateways, and the data which can be in the form of archives, snapshots, reports etc, are recorded at the respective central database. Different IoT gateways and a central database will commit data sharing behaviours. An IT administrator creates and defines an access policy control that is achieved through creating attribute tokens granted for the different entities that interns run the ABE protocol with the predefined access control trees to access the stored data based on their different levels of attributes. Any illegible entity with the required attributes can then connect to the IT infrastructure to acquire the necessary data. This would monitor the data access among the gateways and the database.

ABE is also required by the "Smart Aerospace" use case where the sensors within an aeroplane can share data with the Secure Server Router (SSR) after passing a successful on- boarding process through the ASSURED authentication mechanism. The SSR administrator needs first to define the access policy in a form of smart contract by granting access tokens that regulates the access to ASSURED private ledger. If the authentication is successful and the access policy can be satisfied, the SSR can share its data with a Ground Station Server and the latter can then share data with the analytics cloud Servers.

ABE will also be one of the main ASSURED components adopted by the "Smart Satellites" use case where the Ground Station (GS) is allowed to share threat intelligence data if it processes an access token that enables data sharing. The data access policy should be clearly defined in the smart contracts so that access policies can be checked during data sharing. ASSURED attestation mechanisms should enable CubeSat to perform attestation to the GS. The attestation results can be verified and recorded on an accessible platform (the private ledger) for internal entities. For external entities, ASSURED DLT will enable them to read the attestation data. In this case, access control policy enforcement is required through ABE.

The TPM-based Wallet supports ABE that is mainly used to ensure legitimate attribute-based access control (ABAC) to sensitive encrypted data. A trusted authority which can be many entities (in our context it is the SCB that takes up this role), generates the user attribute keys using the authority master key. The trusted authority then sends the encrypted attribute symmetric decryption keys together with the attribute policies to the edge device which contains an embedded TPM Wallet. The TPM Wallet stores the decryption symmetric keys safely inside the TPM. Using these keys together with the access control tree the TPM can recover the main decryption and integrity keys that will be used by the host to check the message integrity through a message authentication code (MAC) and decrypt a message respectively.

3.3.2 ABE in Searchable Encryption

Indeed, searchable encryption not only protects data privacy of data owners but also enables data users to search over the encrypted data. However, it assumes that the user sending the search query owns the decryption key and that the sender has to know the identity of the user querying the data in order to encrypt using the corresponding encryption key. This raises the question in ASSURED blockchain framework that the encrypted data is shared between several receivers and is kept in a remote shared storage that is not trusted for confidentiality.

To remedy the defects, we can borrow ideas from attribute-based encryption: only participants with the appropriate permissions and the corresponding ABE (matching) attribute private keys can decrypt and view the documents in the off-chain blocks containing the requested keywords. Concretely, the secret decryption key of the encrypted documents containing the requested keywords are related to a set of attributes in some fashion, for which holds that if there is a subset of attributes that consists of at least t attributes that match the set of attributes associated with the secret keys, then the secret keys can be used to decrypt the documents.

As we will see in Chapter 4, our searchable encryption scheme is built between the attestation keywords and the corresponding location pointers to the ASSURED data storage engine where the attestation raw data is stored. Therefore, the encryption of contents stored on the storage engine is completely independent of the searchable encryption scheme. This brings a great benefit: the contents stored on the ASSURED data storage engine can be encrypted using an independent attribute-based encryption (ABE) scheme. For example, we can use the ABE scheme presented in [43], which is based on the Elliptic Curve Integrated Encryption Scheme, in our searchable encryption. A general overview of the aforementioned ABE scheme using the TPM Wallet in ASSURED framework can be found in D4.1 [12, Figure 12]. For a very detailed instantiation of the ABE scheme in ASSURED, we refer to Deliverables 4.1 [12] and 4.2 [17] where they described an ASSURED-designed ABE scheme that allows ASSURED blockchain users to perform ABE decryption using the corresponding keys stored in the TPMs.

3.4 Searchable Encryption

As encrypted data is usually uploaded to remote servers to lighten personal devices' storage pressure and ease the data sharing with geographically demote devices, data retrieval from the server encounters a big challenge. That is, how to request the server for desired data without threatening the data confidentiality. Sensitive data must be preprocessed before submitting to the server to keep confidentiality. Therefore, the server cannot locate data based on content or semantics. The searchable encryption (SE) technologies were proposed to meet the demand of rapid locating desired documents among the massive data in the cloud storage and simultaneously assuring privacy. Basically, SE acts as a data management technique that allows the data owners to store and manage their data at a third-party, untrusted cloud server and allows the data user to delegate search functionality to the cloud server to retrieve that data. Hence, SE enables the secure storage and retrieval of data at the optimized cost. SE has application in the scenario where we want both confidentiality and accessibility. In addition to its direct use in searching encrypted data, SE has also changed the way any encryption technique can work. Generally, an encryption technique either decrypts the complete data or none of it, which makes it unsuitable for situations where only a part of the data needs to be decrypted. To design such a flexible system, the SE technique can be used where one can search a particular part of the encrypted data and thereby can achieve partial decryption.

3.4.1 Need for Searchable Encryption

As defined in the ASSURED conceptual architecture [16], data sharing is one of the cornerstones of the ASSURED framework, as the overall principle for working with secure and trusted devices in a flexible SoS deployment is highly based on the ability to produce and exchange the necessary data between the different entities of the system. ASSURED uses DLTs to build a flexible and expandable data sharing network, because DLT can provide the necessary features that is

necessary to guarantee the trust that is necessary to be available in such a system as identified in D4.1 [12], and the trustworthiness and immutability of any data sharing transaction that needs to take place over an ASSURED deployment.

However, using ledgers has also some drawbacks, which mostly have to do with performance during the execution of operations that need to be performed over the ledgers when it comes to vast amounts of data or the introduction and querying of many records. For this purpose, hybrid approaches are employed in ASSURED, such as having off-chain storage facilities where data (especially the system traces, as monitored during the execution of a remote attestation process, whose size might excess the order of KBytes) is placed and the location of those is provided as pointers which are stored in the ledgers. The off-chain data is stored over a cloud-based data storage engine which may not be trusted with security of sensitive data. As known, data privacy issue are the most prominent and important one when it comes to cloud storage, and could theoretically be easily resolved by storing the data in an encrypted form. However, although encryption solves the problem of privacy, it also engenders some other serious issues including infeasibility of the fundamental search operation, reduction in flexibility of sharing the data with other users etc. To address these issues, the concept of searchable encryption is introduced and developed on ASSURED blockchain ledger. SE allows blockchain ledger peer(s) to perform search on encrypted and stored data on ledger(s) without disclosing any information about what is being searched to the peer(s). Secure SE is the answer to this need.

3.4.2 Security Guarantees

Like the considerations when we measure other cryptosystems' privacy performance, the Searchable Encryption component in ASSURED should request the following security guarantees.

- **Data Privacy.** Searchable encryption depends on a third-party server to receive and store scripts which imply a keyword-document relationship for later data retrieval. That is, the index or the searchable ciphertext is visible to the server and may be exposed to eavesdroppers in the public communication channel on the way to server storage. Data privacy requires that the content of documents should be concealed, even though index or searchable ciphertext is transmitted and stored, i.e., semantic security [3].

- **Keyword Privacy.** Given a search query, i.e., the token or the trapdoor representing the keywords of user's interest, keyword privacy requires that the token or the trapdoor should not leak information about the underlying keyword while used for testing index or searchable ciphertexts. The Keyword Guessing Attack (KGA) [6, 44] is famous for its vulnerability to keyword privacy, exploiting freely manufactured ciphertexts of any keywords to infer the keyword of a trapdoor.

- **Search Pattern Privacy.** Search pattern refers to information about whether two queries are launched for the same keyword [20], which was used to be thought inevitable when most search queries of searchable encryption schemes, especially SSE, were certain and fixed for identical keywords.

- **Access Pattern Privacy.** Access pattern is the document identifiers returned for a search [20]. At the end of the search, the query's document identifiers may be revealed to get search results returned. Such disclosure may further lead to unexpected data compromised [27].

As we will see in Chapter 4, the attestation raw data is encrypted under an ABE scheme and stored on the ASSURED data storage engine. The searchable encryption scheme deployed in ASSURED (see Chapter 4) is built between the attestation keywords and the corresponding location pointers to the ASSURED data storage engine where the attestation raw data is stored. The encryption of the attestation raw data is completely independent of the SSE scheme. Furthermore, the location pointers are also encrypted using a symmetric encryption scheme (e.g. AES-CBC), thus data privacy is guaranteed.

We have mentioned briefly before that the SCB will use the random oracles to generate random strings related to the keyword w, and then mask the identifiers and the storage location information of documents containing the keyword w. When adding/deleting a document, the SCB also uses the random oracles to generate random strings to mask the corresponding addition/deletion tokens. Therefore, when searching keywords (or adding/deleting documents) on the public ledger, the keywords are not in plaintext and the related tokens will not leak information about the underlying keywords while used for testing index or searchable ciphertexts. However, a limitation of all known SSE constructions (including the one employed in ASSURED) is that the tokens they generate are deterministic, in the sense that the same token will always be generated for the same keyword. This means that searches leak statistical information about the user's search pattern. Currently, it is unknown how to design efficient SSE schemes with probabilistic trapdoors. Fortunately, the security model in the SSE scheme employed in ASSURED is different from those in literature, mainly in that all the tokens (including search, addition and deletion) are generated by the SCB which is assumed to be honest and trusted comparing with the server (in literature) that is honest but curious. So the search pattern privacy is also guaranteed in our (dynamic) SSE scheme.

All existing schemes in the literature leak the access pattern. At the current stage, it is unknown how to design efficient SSE schemes that can protect access pattern. Thus, we won't consider the access pattern privacy in our design as well. However, as mentioned above, we actually have a different security model that all the privacy-related operations are performed by an honest and trusted party, i.e., the Security Context Broker (SCB). So, we expect that we can obtain the privacy guarantee of access pattern to some extent.

Captured Security Requirements: DLT-SCT-03, DLT-SCT-05, DLT-SCT-07, DLT-SCT-08, DLT-SCT-13, DLT-SCT-14, DLT-SCT-17, DLT-SCT-21, DLT-SCT-22, DLT-SCT-25, DLT-SCT-34, DLT-SCT-35, DLT-SCT-36, DLT-SCT-38, DLT-SCT-39, DLT-SCT-45, DLT-SCT-46, DLT-SCT-47, DLT-DEC-06, DLT-SHA-14.

3.4.3 General Model

Though the searchable encryption technologies are categorized into two classes, i.e., Searchable Symmetric Encryption (SSE) and Public-key Encryption with Keyword Search (PEKS), their four algorithms are similar in syntax.

- **Setup/Key Generation.** In this algorithm, system parameters are set, and keys are generated for each party.
- **Build Index/Encrypt.** In this algorithm, the data owner processes the keywords of their documents and uploads this script, i.e., index or searchable ciphertext, to the server.
- **Token/Trapdoor Generation.** In this algorithm, the authorized data user computes the token or trapdoor based on the keyword of its interest, which is used for later testing with an index item or a searchable ciphertext.

- **Search/Test.** In this algorithm, the server validates the received token or trapdoor with the index item or searchable ciphertext in storage to decide to return the corresponding document or not.

There are few schemes that implement searchable encryption by making the ciphertext itself searchable [42]; most of the schemes generate an encrypted index which is searchable. The searchable encrypted index can be generated by extracting the metadata items $\boldsymbol{w} = (w_1, \ldots, w_m)$ from the main data file and encrypting them using a technique which enables search operation over this index. This operation is performed by executing the encryption algorithm defined in the searchable encryption scheme. These metadata items are often called the keywords in searchable encryption terminology. The main data file is also encrypted with any standard symmetric-key/asymmetric-key encryption. The symmetric key used for this purpose may be encrypted with the same encryption technique which we used for generating the index.

The encrypted index and the encrypted data file are then stored on a cloud server (the data storage engine in ASSURED) that is generally assumed to be honest but curious, i.e. it can be trusted to correctly perform storage and query protocols, however, it tries to learn as much information as possible. This whole task is done by an entity named the Security Context Broker (SCB) in ASSURED. Now, another entity called the data user wants to retrieve the encrypted information stored at the cloud server. The user needs to first search among the data files to retrieve what he/she needs. The data user delegates this search operation to the cloud server by generating a search trapdoor or sometimes called a search token for the keyword he/she is looking for, by using the trapdoor algorithm defined in the searchable encryption scheme (Trapdoors) and sends it to the cloud server. Using the search trapdoor, the cloud server is now able to perform search in the encrypted domain and returns the reference to the file which contains the queried keyword without actually decrypting anything. Figure 3.1 gives a general model of an index-based scheme.

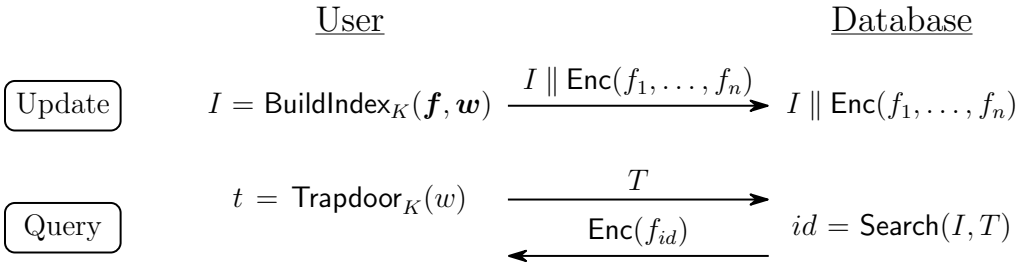

Figure 3.1: General Model of an Index-Based Searchable Encryption Scheme

To generate the index, generally there are two approaches, the forward approach,

document id	keywords
f_1	$w_{i_1}, w_{i_2}, w_{i_3} \ldots$
f_2	$w_{j_1}, w_{j_2}, w_{j_3}, \ldots$
$\ldots$	$\ldots$
f_n	$w_{n_1}, w_{n_2}, w_{n_3}, \ldots$

and the inverted approach,

keyword	document ids
w_1	$f_{k_1}, f_{k_2}, f_{k_3} \ldots$
w_2	$f_{l_1}, f_{l_2}, f_{l_3}, \ldots$
$\ldots$	$\ldots$
w_m	$f_{m_1}, f_{m_2}, f_{m_3}, \ldots$

In the forward approach, an index is built over the data files $\boldsymbol{f} = (f_1, \ldots, f_n)$, i.e. to each data file some keywords are associated. In the inverted approach, an index is built over the keywords $\boldsymbol{w} = (w_1, \ldots, w_m)$, i.e. it indexes each keyword belongs to that data file.

In the forward-index approach, the search time varies linearly with the number of data file. In the inverted-index approach, the search time varies with the number of documents containing a particular keyword and it can be reduced to sublinear if a key-value dictionary approach, such as a hash table/tree, is used to build the index. One of the approaches is followed while developing a searchable encryption technique but generally it is not shown explicitly in the construction. The only focus of a searchable encryption scheme is to encrypt the associated metadata (keywords) with the data file in an efficient way.

Due to the efficiency and scalability issue, we will not make use of public key-based SE but only symmetric key-based SE. For searchable symmetric encryption component in ASSURED, we will investigate the inverted-index approach in the design and implementation stages since it allows a sublinear search complexity as mentioned above. Basically, ASSURED will use SSE technique in data query over the on-chain structures. The on-chain structures may store the index structure related to the data/information along with their off-chain and/or database location. An entity that performs a query may just provide a keyword-hidden token to the admin who manages the on-chain structures. Then, an encrypted location information, related to off-chain data storage engine, can be returned to that entity for decryption. In this way, the admin cannot see what data the entity is currently searching. Similar technique can be also used in backend database storage for privacy-preserving database data query.

3.4.4 Searchable Symmetric Encryption

As introduced in Section 3.1, symmetric encryption allows a single user to read and write data, and allows only the secret key holder to create searchable ciphertexts and trapdoors [46]. The immediate application of searchable symmetric encryption (SSE) can be seen in the scenario where some user wants to store his/her data on the cloud storage in an encrypted form and later he/she can perform search on it to retrieve the required data [22]. The index approach is explicitly mentioned and addressed in the SSE schemes only. The first SSE was proposed by Song et al. [36]. The proposed scheme supports equality query and was developed to handle just one keyword. To encrypt the data, first they break it into fixed size blocks called words and then each word is encrypted independently by inserting a hash value in the specific format. The ciphertext resulting from this process is itself searchable. The cloud server performs search by extracting the hash and looking for the specific format indicated by the user. However, this scheme has a restriction that it needs to break the data into fixed size words which are incompatible with the existing encryption standard. Further, they did not provide any formal security definition or security proof for their scheme. Later, in 2003, Goh [24] relaxed the fixed size restriction in the Song et al. scheme. They used a direct index approach where an index is built for every data file using Bloom filters. In 2005, Chang and Mitzenmacher [10] developed an SSE scheme and, like the Goh scheme, they used a direct index approach where an index is built based on

the dictionary of search keywords instead of Bloom filters, which inherently have very high false positive rate. In 2006, Curtmola et al. [20] for the first time used an inverted-index approach and achieved the sublinear search time.

Subsequent researches [7, 8, 11, 28–30, 32, 33] start to extract an auxiliary index so that there is no need to scan the whole document and thus search is accelerated. Security definitions were correspondingly formalized to depict the security requirements for SSE, e.g., semantic security against adaptive Chosen Keyword Attack (IND-CKA) [24]. The scheme in [24] was later pointed out by [10] leaking information like the length of "1" entries in the Bloom filter and incurs an enhanced version on its security definition followed by a modified scheme. Later on, simulation-based security models, including non-adaptive indistinguishability and adaptive indistinguishability [20] have been presented to outline better the security requirements for SSE, followed by concrete schemes. Kurosawa et al. [32] later pointed out the second scheme of [20] has design flaws and fails to meet their claimed security. In such a tortuous journey, Searchable Symmetric Encryption schemes, as well as corresponding security models, have been continuously investigated.

3.4.5 Dynamic Searchable Symmetric Encryption

An important challenge for the searchable encryption community has been the development of dynamic schemes. As large-scale data storage always receives many data update requests, a dynamic searchable symmetric encryption (DSSE) that supports efficient data or keyword modification via an update mechanism better meets the user requirements. Until the work of Kamara, Papamanthou and Roeder [30], there was no efficient dynamic searchable encryption scheme. By 'efficient', we mean a scheme whose search complexity is sublinear in the number of documents, whose encrypted database size is linear in the size of the plain dataset, and whose update complexity is linear in the number of updated document/keyword pairs. The scheme of Kamara et al. is based on the encrypted-linked-list-based multimap of [19, 20] and performs insertions by (logically) enqueuing each new entry in the right list. Efficient deletions are supported by encoding a dual representation of the index: also using linked lists, the client can easily tell the server which tokens of the encrypted database correspond to a given document, and hence tell him to remove them when needed.

Unfortunately, a dynamic SSE always encounters more challenges than a static one. For example, dynamism allows for new means of attack, and regular dynamic index-based schemes, such as the ones in [7], are subject to devastating adaptive attacks [45]. To thwart these attacks, forward-private schemes have been constructed [4, 38], i.e. schemes whose update request should not reveal anything about queried keywords before. Also, for most dynamic schemes, a deletion only logically removes the entries, but the server is still able to decrypt these deleted entries—and hence extract some supposedly erased information—for any subsequent search request matching these. However, this problem can be fixed using backward private constructions [5]. The backward security means that the queries cannot be performed over deleted documents. Some formalized work by Stefanov et al. [38] and Bost et al. [4, 5], have shifted efforts to develop dynamic SSE schemes with forward and backward security.

In 2016, Garg et al. constructed a forward-secure DSSE scheme based on their TWORAM [23]. In 2017, Kim et al. utilized the dual dictionary to construct a forward-secure DSSE scheme that supports real deletion [31]. In 2018, Song et al. proposed a counter-based forward-secure DSSE scheme FAST/FASTIO with real deletion support [37]. Their proposed scheme achieves I/O efficiency by caching historical search results.

In 2016, Hoang et al. presented forward-and-backward-secure DOD-DSSE [26]. The core idea of DOD-DSSE is to let the client fetch all related data from the server and to perform Search or Update operations locally. In 2017, Bost et al. [5] constructed four forward-and-backward-secure DSSE schemes: Fides, $\text{Diana}_{\text{del}}$, Moneta and Janus. Subsequently, Chamani et al. [9] proposed three improved constructions, including Type-I scheme Orion, Type-II scheme Mitra and Type-III scheme Horus. At the same time, Sun et al. [40] proposed a practical Type-III scheme Janus++ by making use of their symmetric puncturable encryption. In 2019, Li et al. constructed a forward-and-backward-secure DSSE Khons with the hidden pointer ciphertext structure and partition search technique [34]. In 2020, He et al. [25] presented a forward-and-backward-secure DSSE scheme CLOSE-F/CLOSE-FB with constant client storage. He et al.'s approach is to combine the counter and chain structure and use the global counter to find all chain structures of ciphertexts.

In the same year, Demertzis et al. [21] proposed three forward-and-backward-secure DSSE scheme, QOS, SD_a and SD_d with constant client storage. The first two schemes in [21] achieve interactive real deletion, and the third scheme uses oblivious map and a tree-based encrypted index as building blocks. Amjad et al. [1] proposed several schemes with all types of backward privacy by leveraging the power of Intel SGX. They are all non-interactive but depending on the security and reliability of trusted execution environments. Very recently in 2021, Sun et al. [39] proposed a forward-and-backward-secure DSSE scheme Aura, which achieves non-interactive real deletion in the cost of extra client-side storage resources to stash Delete queries.

In the ASSURED project, **we will prefer to make the research work [30] as our starting point. This is because their construction provides high search efficiency and low client storage cost.** In Chapter 4, we will see how to leverage the features of it and to design and develop a dynamic SSE scheme to maintain strong search privacy and data security but also highly scalability and practicality while merging the dynamic SSE with our ASSURED blockchain framework, in the public ledger's secure metadata search. Intuitively, we will enable the SCB to construct keyword-based index structure for attestation metadata and the corresponding attestation reports' storage pointers, and this structure can be further dynamically expanded as the increase of reports. The structure and the pointers will be further encrypted and stored on an ASSURED public ledger. For each external party who is trying to search over the encryption, the SCB will construct a search token corresponding to the given keyword, and with the token, the public ledger peer(s) can locate the required encrypted pointers. Then the external party can use the pointer to retrieval the encrypted attestation raw data from the data storage engine. Then TPM-enabled ABE scheme is mainly used to ensure legitimate attribute-based access control to sensitive encrypted data. In our ASSURED framework, blockchain users will send the encrypted attribute symmetric decryption keys together with the attribute policies to the edge device which contains an embedded TPM Wallet. The TPM Wallet stores the decryption symmetric keys safely inside the TPM. Using these keys together with the access control tree, if the host has matching attributes, the TPM can recover the main decryption key that will be used by the host to decrypt the off-chain attestation metadata. Note more details about interaction among DSSE keys and TPM, please refer to the deliverable D4.5 [18].

Chapter 4

ASSURED Initial Designed Dynamic Searchable Symmetric Encryption Scheme

4.1 Notations

The set of all binary strings of length n is denoted as $\{0,1\}^n$, and the set of all finite binary strings as $\{0,1\}^*$. The notation $[n]$ represents the set of integers $\{1,\ldots,n\}$. We write $x \leftarrow \chi$ to represent an element x being sampled from a distribution χ, and $x \leftarrow\!\!\$\, X$ to represent an element x being sampled uniformly at random from a set X. The output x of a probabilistic algorithm $\mathcal{A}$ is denoted by $x \leftarrow \mathcal{A}$ and that of a deterministic algorithm $\mathcal{B}$ by $x := \mathcal{B}$. Given a sequence of elements $\mathbf{v}$ we refer to its ith element either as v_i or $\mathbf{v}[i]$ and to its total number of elements as $\#\mathbf{v}$. If S is a set then $\#S$ refers to its cardinality. W denotes the universe of words. If $f = (w_1,\ldots,w_m) \in W^m$ is a file, then $\#f$ denotes its total number of words and $|f|$ is its bit length. Also, $\bar{f}$ is the file that results from removing all duplicates from f (i.e., $\bar{f}$ contains only the unique words in f sequenced according to the order in which they first appear in f). If s is a string then $|s|$ refers to its bit length. We denote the concatenation of n strings $s_1,\ldots,s_n$ by $\langle s_1,\ldots,s_n\rangle$. We denote by $\perp$ the special value to signify an error or a failure in the execution of the protocol.

We use various data structures including linked lists, arrays and dictionaries. If $\mathtt{L}$ is a list then $\#\mathtt{L}$ denotes its total number of nodes. If $\mathtt{A}$ is an array then $\#\mathtt{A}$ is its total number of cells, $\mathtt{A}[i]$ is the value stored at location $i \in [\#\mathtt{A}]$ and $\mathtt{A}[i] := v$ denotes the operation that stores v at location i in $\mathtt{A}$. A dictionary (also known as a key-value store or associative array) is a data structure that stores key-value pairs (s,v). If the pair (s,v) is in $\mathtt{T}$, then $\mathtt{T}[s]$ is the value v associated with s. $\mathtt{T}[s] := v$ denotes the operation that stores the value v under search key s in $\mathtt{T}$ and $\#\mathtt{T}$ is the number of pairs in $\mathtt{T}$. We sometimes write $s \in \mathtt{T}$ to mean that there exists some pair in $\mathtt{T}$ with search key s.

4.2 System Model

We employ a multi-party model mainly including the following four parties: a trusted data owner, a trusted SCB, a semi-trusted ASSURED data storage engine, and a collection of users who are authorized to search. The task for each party is described as follows:

- Data owner: An authorized data owner would like to outsource a collection of documents $\mathbf{f} = (f_1,\ldots,f_n)$ together with some keywords $\mathbf{w} = (w_1,\ldots,w_m)$. In ASSURED context, we assume that the devices who generate the attestation data are the data owners.

- Blockchain Peers: The main functions the peers can provide are: send the attestation bundle, including attestation results, attestation ID, encrypted attestation raw data, and related metadata to the SCB; perform secure share over the public ledger; and help private channel users to access the attestation results stored on the private ledger.
- ASSURED Ledgers: The public ledger stores the encrypted index structure and encrypted pointers; and the private ledger is used to store attestation results and related information.
- SCB: After obtaining the attestation bundle from the peers, it forwards the encrypted data to the data storage engine, and then receives the corresponding pointer. It makes use of searchable encryption component to construct encrypted index structure and encrypted pointer, and further stores them on the public ledger. The SCB also performs search/update token generation tasks. When the SCB receives a query keyword from an external party, it constructs a search token so that the peer can search over the public ledger and then returns related encrypted pointer. The SCB can further decrypt the pointer for the external party. We here assume that the SCB is honest and trusted. This means it will follow the protocol correctly.
- ASSURED Data Storage Engine: The off-chain cloud-based storage engine stores the encrypted data under an ABE scheme, and responds to the SCB's (and external parties with granted rights) requests and queries. The storage engine is relied on to provide highly-available and reliable storage.
- Data user: If an external party wants to search the data that contains a particular keyword, he/she has to submit this query keyword to the SCB. After searching, the SCB returns the encrypted data that contains this keyword to the user. As for internal private channel users, e.g., a device, it can directly request the private ledger peer to return the attestation result and plaintext pointer. In the ASSURED context, the external and internal parties can be both regarded as data user.

A considerable amount of attestation data can be viewed as a sequence of n files $\mathbf{f} = (f_1, \ldots, f_n)$, where file f_i is a sequence of words $(w_1, \ldots, w_m)$ from a universe W. We assume that each file (i.e. each attestation file) has a unique identifier $\mathsf{id}(f_i)$, which we regard this as a pointer, e.g., `_uid`, which will be further defined in Chapter 5. We further note that this identifier is distinct and different from the attestation ID, and it is only used to link its storage location back to the data storage engine. We also assume that all the attestation data can be dynamic, so at any time an attestation file may be added or removed. We note that the files do not have to be text files but can be any type of data as long as there exists an efficient algorithm that maps a given document to a file of keywords from W. Given a keyword w we denote by $\mathbf{f}_w$ the set of files in $\mathbf{f}$ that contain w. If $\mathbf{c} = (c_1, \ldots, c_n)$ is a set of encryptions of the files in $\mathbf{f}$, then $\mathbf{c}_w$ refers to the ciphertexts that are encryptions of the files in $\mathbf{f}_w$.

Definition 2 (Dynamic SSE)**.** *A dynamic index-based SSE scheme is a tuple of nine polynomial-time algorithms* $\mathsf{SSE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{SrchToken}, \mathsf{AddToken}, \mathsf{DelToken}, \mathsf{Search}, \mathsf{Add}, \mathsf{Del}, \mathsf{Dec})$ *such that:*

- $K \leftarrow \mathsf{Gen}(1^\lambda)$*: is a probabilistic algorithm that takes as input a security parameter* λ *and outputs a secret key* K*.*
- $(\gamma, \mathbf{c}) \leftarrow \mathsf{Enc}(K, \mathbf{f})$*: is a probabilistic algorithm that takes as input a secret key* K *and a sequence of files* $\mathbf{f}$*. It outputs an encrypted index* γ*, and a sequence of ciphertexts* $\mathbf{c}$*.*

- $\tau_s \leftarrow \mathsf{SrchToken}(K, w)$*: is a (possibly probabilistic) algorithm that takes as input a secret key* K *and a keyword* w*. It outputs a search token* τ_s*.*
- $(\tau_a, c_f) \leftarrow \mathsf{AddToken}(K, f)$*: is a (possibly probabilistic) algorithm that takes as input a secret key* K *and a file* f*. It outputs an add token* τ_a *and a ciphertext* c_f*.*
- $\tau_d \leftarrow \mathsf{DelToken}(K, f)$*: is a (possibly probabilistic) algorithm that takes as input a secret key* K *and a file* f*. It outputs a delete token* τ_d*.*
- $\mathbf{I}_w := \mathsf{Search}(\gamma, \mathbf{c}, \tau_s)$*: is a deterministic algorithm that takes as input an encrypted index* γ*, a sequence of ciphertexts* $\mathbf{c}$ *and a search token* τ_s*. It outputs a sequence of identifiers* $\mathbf{I}_w \subseteq \mathbf{c}$*.*
- $(\gamma', \mathbf{c}') := \mathsf{Add}(\gamma, \mathbf{c}, \tau_a, c)$*: is a deterministic algorithm that takes as input an encrypted index* γ*, a sequence of ciphertexts* $\mathbf{c}$*, an add token* τ_a *and a ciphertext* c*. It outputs a new encrypted index* γ' *and sequence of ciphertexts* $\mathbf{c}'$*.*
- $(\gamma', \mathbf{c}') := \mathsf{Del}(\gamma, \mathbf{c}, \tau_d)$*: is a deterministic algorithm that takes as input an encrypted index* γ*, a sequence of ciphertexts* $\mathbf{c}$*, and a delete token* τ_d*. It outputs a new encrypted index* γ' *and new sequence of ciphertexts* $\mathbf{c}'$*.*
- $f := \mathsf{Dec}(K, c)$*: is a deterministic algorithm that takes as input a secret key* K *and a ciphertext* c *and outputs a file* f*.*

4.3 Dynamic Searchable Symmetric Encryption Scheme

4.3.1 A High Level Description

In this chapter we will mainly introduce a detailed dynamic SSE scheme used in ASSURED. In what follows, we will first provide a high level description of the dynamic SSE, and more details can be found in the follow-up sections.

Let $\lambda \in \mathbb{N}$ be the security parameter, $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an anonymous and CPA-secure symmetric encryption scheme and $F : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^\lambda$, $G : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^*$, and $P : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^\lambda$ be pseudorandom functions. Let $H_1 : \{0,1\}^* \rightarrow \{0,1\}^*$ and $H_2 : \{0,1\}^* \rightarrow \{0,1\}^*$ be random oracles.

To set up the encrypted database, the SCB first samples three λ-bit strings K_1, K_2, K_3 uniformly at random and generates $K_4 \leftarrow \mathsf{SKE.Gen}(1^\lambda)$. The collection of files $\mathbf{f}$ is then encrypted using the symmetric encryption scheme SKE to obtain a sequence of ciphertexts $\mathbf{c} = (\mathsf{SKE.Enc}_{K_4}(f_1), \ldots, \mathsf{SKE.Enc}_{K_4}(f_{\#\mathbf{f}}))$.

To build the index, for each keyword $w \in W$ we construct a list $\mathtt{L}_w$. Each list $\mathtt{L}_w$ is composed of $\#\mathbf{f}_w$ nodes $(\mathtt{N}_1, \ldots, \mathtt{N}_{\#\mathbf{f}_w})$ that are stored at random locations in the search array $\mathtt{A}_s$. The node $\mathtt{N}_i$ is defined as $\mathtt{N}_i = \langle \mathsf{id}, \mathsf{addr}_s(\mathtt{N}_{i+1}) \rangle$, where id is the unique file identifier of a file that contains w and $\mathsf{addr}_s(\mathtt{N})$ denotes the location of node $\mathtt{N}$ in $\mathtt{A}_s$. Here the above process is constructed based on the inverted approach as introduced in the previous chapter. And this process is completed by the SCB. Before this process, the SCB has to extract all the necessary keywords from the metedata received from the peers, which are related to the attestation data. After the keywords are extracted, the SCB will link each of them to all the existing attestation data, for example, a keyword w_1 is associated with attestation files f_1 and f_2, in which f_1 and f_2 are seen as pointers

(referring to the data storage engine). Once the process is done, the SCB will have a group of keywords to form a keyword dictionary. This dictionary will be stored on the public ledger, so that external parties can understand which keywords they can use for queries. We note that at this stage, we do not fully define concrete keywords, e.g., "Monday", "attestation time", "device's name". A concrete generation for all the keywords will be considered in the implementation stage. And this will not affect the current design, because we currently define an abstract and generic construction for these keywords and later concrete values can be directly adopted in this design.

For each keyword w, an index to the head of $\mathtt{L}_w$ is then inserted into the search table $\mathtt{T}_s$ under search key $F_{K_1}(w)$, where K_1 is the key to the PRF F. Each list is then encrypted using SKE under a key generated as $G_{K_2}(w)$, where K_2 is the key to the PRF G.

To search for a keyword w, it suffices for the requesting entity to send the values $F_{K_1}(w)$ and $G_{K_2}(w)$. The server can then use $F_{K_1}(w)$ with $\mathtt{T}_s$ to recover the index to the head of $\mathtt{L}_w$, and use $G_{K_2}(w)$ to decrypt the list and recover the identifiers of the files that contain w. As long as $\mathtt{T}$ supports $O(1)$ lookups (which can be achieved using a hash table), the total search time for the server is linear in $\#\mathbf{f}_w$, which is optimal since at a minimum the server needs to fetch the relevant documents just to return them.

To support efficient dynamic update, we add an extra (encrypted) data structure $\mathtt{A}_d$ called the deletion array that the ledger can query (with a token provided by the SCB) to recover pointers to the nodes that correspond to the file being deleted. More precisely, the deletion array stores for each file f a list $\mathtt{L}_f$ of nodes that point to the nodes in $\mathtt{A}_s$ that should be deleted if file f is ever removed. So every node in the search array has a corresponding node in the deletion array and every node in the deletion array points to a node in the search array. Throughout, we will refer to such nodes as duals and write $\mathtt{N}^*$ to refer to the dual of a node $\mathtt{N}$.

The encrypted pointer is stored in a node with a homomorphic encryption scheme. This is similar to the approach used by van Liesdonk et al in [41] to modify the encrypted search structure they construct. By providing the ledger with an encryption of an appropriate value, it can then modify the pointer without ever having to decrypt the node. We use the "standard" private-key encryption scheme which consists of XORing the message with the output of a PRF. This simple construction also has the advantage of being non-committing (in the private-key setting) which we make use of to achieve CKA2-security.

4.3.2 A Detailed Description

The ASSURED designed dynamic SSE scheme $\mathsf{SSE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{SrchToken}, \mathsf{AddToken}, \mathsf{DelToken}, \mathsf{Search}, \mathsf{Add}, \mathsf{Del}, \mathsf{Dec})$ is described in detail as the following.

Setup. Let $\lambda \in \mathbb{N}$ be the security parameter, $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an anonymous and CPA-secure symmetric encryption scheme and $F : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda$, $G : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^*$, and $P : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda$ be pseudo-random functions. Let $H_1 : \{0,1\}^* \to \{0,1\}^*$ and $H_2 : \{0,1\}^* \to \{0,1\}^*$ be random oracles. Setting the keywords universe W, which is a finite set of all possible keywords that occur in all files (including the files added in the future) and are represented by binary strings of arbitrary finite length.

- $\mathsf{Gen}(1^\lambda)$: Sample three λ-bit strings K_1, K_2, K_3 uniformly at random and generate $K_4 \leftarrow \mathsf{SKE.Gen}(1^\lambda)$. Output $K = (K_1, K_2, K_3, K_4)$.

Building the index structure.

- $\mathsf{Enc}(K, \mathbf{f})$:

  1. For $1 \leq i \leq \#\mathbf{f}$, let $c_i \leftarrow \mathsf{SKE.Enc}_{K_4}(f_i)$. Set $\mathbf{c} = (c_1, \ldots, c_{\#\mathbf{f}})$.
  2. Let $\mathtt{A}_s$ and $\mathtt{A}_d$ be arrays of sufficiently large size and let $\mathtt{T}_s$ and $\mathtt{T}_s$ be dictionary of size $\#W$ and $\#\mathbf{f}$, respectively. Denote by $\mathsf{len}_{\mathsf{addr}}$ the size of the address of a cell in $\mathtt{A}_s$, and by $\mathsf{len}_{\mathsf{id}}$ the size of the identifier of a file. We assume $\mathbf{0}$ is a $\mathsf{len}_{\mathsf{addr}}$-length string of 0's.
  3. For each word $w \in W$: (**Steps 3 and 4 here must be performed in an interleaved fashion to set up $\mathtt{A}_s$ and $\mathtt{A}_d$ at the same time.**)
     (a) Create a list $\mathtt{L}_w$ of $\#\mathbf{f}$ nodes $(\mathtt{N}_1, \ldots, \mathtt{N}_{\#\mathbf{f}})$ stored at random locations in the search array $\mathtt{A}_s$ and defined as:

     $$\mathtt{N}_i := \left(\left\langle \mathsf{id}_i, \mathsf{addr}_s(\mathtt{N}_{i+1}) \right\rangle \oplus H_1(K_w, r_i), r_i\right)$$

     where id_i is the file identifier of the ith file in $\mathbf{f}_w$, $\mathsf{addr}_s(\mathtt{N}_j)$ is the address of node $\mathtt{N}_j$ in the search array $\mathtt{A}_s$, r_i is a λ-bit string generated uniformly at random, $K_w := P_{K_3}(w)$ and $\mathsf{addr}_s(\mathtt{N}_{\#\mathbf{f}_w+1}) = \mathbf{0}$.
     (b) Store a pointer to the first node of $\mathtt{L}_w$ in the search table by setting

     $$\mathtt{T}_s[F_{K_1}(w)] := \left\langle \mathsf{addr}_s(\mathtt{N}_1), \mathsf{addr}_d(\mathtt{N}_1^*) \right\rangle \oplus G_{K_2}(w),$$

     where $\mathtt{N}_i^*$ is the dual of $\mathtt{N}_i$, i.e., the node in $\mathtt{A}_d$ whose fourth entry points to $\mathtt{N}_i$ in $\mathtt{A}_s$, and $\mathsf{addr}_d(\mathtt{N}_i^*)$ is the address of node $\mathtt{N}_i^*$ in the deletion array $\mathtt{A}_d$.
  4. For each file f in $\mathbf{f}$: (**Steps 3 and 4 here must be performed in an interleaved fashion to set up $\mathtt{A}_s$ and $\mathtt{A}_d$ at the same time.**)
     (a) Create a list $\mathtt{L}_f$ of $\#\bar{f}$ dual nodes $(\mathtt{D}_1, \ldots, \mathtt{D}_{\#\bar{f}})$ stored at random locations in the deletion array $\mathtt{A}_d$ and defined as follows: each entry $\mathtt{D}_i$ is associated with a word w, and hence a node $\mathtt{N}$ in $\mathtt{L}_w$. Let $\mathtt{N}_{+1}$ be the node following $\mathtt{N}$ in $\mathtt{L}_w$, and $\mathtt{N}_{-1}$ the node previous to $\mathtt{N}$ in $\mathtt{L}_w$. Then, define $\mathtt{D}_i$ as follows:

     $$\mathtt{D}_i := \Bigg(\Big\langle \mathsf{addr}_d(\mathtt{D}_{i+1}), \mathsf{addr}_d(\mathtt{N}_{-1}^*), \mathsf{addr}_d(\mathtt{N}_{+1}^*), \mathsf{addr}_s(\mathtt{N}), \mathsf{addr}_s(\mathtt{N}_{-1}), \mathsf{addr}_s(\mathtt{N}_{+1}), F_{K_1}(w) \Big\rangle \oplus H_2(K_f, r_i'), r_i' \Bigg)$$

     where r_i' is a λ-bit string generated uniformly at random, $K_f := P_{K_3}(f)$, and $\mathsf{addr}_d(\mathtt{D}_{\#\bar{f}+1}) = \mathbf{0}$.
     (b) Store a pointer to the first node of $\mathtt{L}_f$ in the deletion table by setting:

     $$\mathtt{T}_d[F_{K_1}(f)] := \mathsf{addr}_d(\mathtt{D}_1) \oplus G_{K_2}(f).$$

  5. Output $(\gamma, \mathbf{c})$, where $\gamma := (\mathtt{A}_s, \mathtt{T}_s, \mathtt{A}_d, \mathtt{T}_d)$.

Keyword search.

- $\mathsf{SrchToken}(K, w)$: Compute and output $\tau_s := (F_{K_1}(w), G_{K_2}(w), P_{K_3}(w))$.
- $\mathsf{Search}(\gamma, \mathbf{c}, \tau_s)$:

1. Parse τ_s as (τ_1, τ_2, τ_3) and return an empty list if τ_1 is not present in $\mathtt{T}_s$.
2. Recover a pointer to the first node of the list by computing $\langle \alpha_1, \alpha_1' \rangle := \mathtt{T}_s[\tau_1] \oplus \tau_2$.
3. Look up $\mathtt{N}_1 := \mathtt{A}[\alpha_1]$ and decrypt with τ_3, i.e., parse $\mathtt{N}_1$ as (ν_1, r_1) and compute
$$\langle \mathsf{id}_1, \mathsf{addr}_s(\mathtt{N}_2) \rangle := \nu_1 \oplus H_1(\tau_3, r_1).$$
4. For $i \geq 2$, decrypt node $\mathtt{N}_i$ as above until $\alpha_{i+1} = \mathbf{0}$.
5. Let $I = \{\mathsf{id}_1, \ldots, \mathsf{id}_m\}$ be the file identifies revealed in the previous steps and output $\{c_i\}_{i \in I}$, i.e., the encryptions of the files whose identifiers were revealed.

File addition.

- $\mathsf{AddToken}(K, f)$:
  1. Let $(w_1, \ldots, w_{\#\bar{f}})$ be the unique words in f in their order of appearance in f. Compute
  $$\phi_a := (F_{K_1}(f), G_{K_2}(f), \pi_1, \ldots, \pi_{\#\bar{f}}),$$
  where for all $1 \leq i \leq \#\bar{f}$:
  $$\pi_i := \Big(F_{K_1}(w_i), G_{K_2}(w_i), \langle \mathsf{id}(f), \mathbf{0} \rangle \oplus H_1(P_{K_3}(w_i), r_i), r_i, \\ \langle \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, F_{K_1}(w_i) \rangle \oplus H_2(P_{K_3}(f), r_i'), r_i' \Big),$$
  and r_i, r_i' are λ-bit strings generated uniformly at random.
  2. Compute $c_f \leftarrow \mathsf{SKE.Enc}_{K_4}(f)$ and output $\tau_a := (\phi_a, c_f)$.
- $\mathsf{Add}(\gamma, \mathbf{c}, \tau_a)$:
  1. Parse τ_a as $((\tau_1, \tau_2, \pi_1, \ldots, \pi_{\#\bar{f}}), c)$ and return $\perp$ if τ_1 is not in $\mathtt{T}_d$.
  2. Find $\#\bar{f}$ new and free locations $(\varphi_1, \ldots, \varphi_{\#\bar{f}})$ in the search array $\mathtt{A}_s$, and $\#\bar{f}$ new and free locations $(\varphi_1^*, \ldots, \varphi_{\#\bar{f}}^*)$ in the deletion array $\mathtt{A}_d$. Let $\varphi_{\#\bar{f}+1}^* = \mathbf{0}$.
  3. For $1 \leq i \leq \#\bar{f}$:
     (a) Recover a pointer to the first node $\mathtt{N}_1$ of the list by computing $\langle \alpha_1, \alpha_1^* \rangle := \mathtt{T}_s[\pi_i[1]] \oplus \pi_i[2]$.
     (b) Sore a new node at location φ_i and modify its forward pointer to $\mathtt{N}_1$ by setting $\mathtt{A}_s[\varphi_i] := \big(\pi_i[3] \oplus \langle 0^{\mathsf{len}_{\mathsf{id}}}, \alpha_1 \rangle, \pi_i[4]\big)$.
     (c) Update the search table by setting $\mathtt{T}_s[\pi_i[1]] := \langle \varphi_i, \varphi_i^* \rangle \oplus \pi_i[2]$.
     (d) Update the dual of $\mathtt{N}_1$ by setting $\mathtt{A}_d[\alpha_1^*] := \Big(\tilde{\mathtt{D}}_1 \oplus \langle \mathbf{0}, \varphi_i^*, \mathbf{0}, \mathbf{0}, \varphi_i, \mathbf{0}, 0^{|\pi_i[1]|} \rangle, r\Big)$, where $(\tilde{\mathtt{D}}_1, r) := \mathtt{A}_d[\alpha_1^*]$.
     (e) Update the dual of $\mathtt{A}_s[\varphi_i]$ by setting $\mathtt{A}_d[\varphi_i^*] := \Big(\pi_i[5] \oplus \langle \varphi_{i+1}^*, \mathbf{0}, \alpha_1^*, \varphi_i, \mathbf{0}, \alpha_1, \pi_i[1] \rangle, \pi_i[6]\Big)$.
     (f) If $i = 1$, update the deletion table by setting $\mathtt{T}_d[\tau_1] := \varphi_1^* \oplus \tau_2$.
  4. Update the ciphertexts by adding c to $\mathbf{c}$.

File deletion.

- DelToken(K, f): Output $\tau_d := (F_{K_1}(f), G_{K_2}(f), P_{K_3}(f), \mathsf{id}(f))$.
- Del$(\gamma, \mathbf{c}, \tau_d)$:
  1. Parse τ_d as $(\tau_1, \tau_2, \tau_3, \mathsf{id})$ and return $\perp$ if τ_1 is not in $\mathtt{T}_d$.
  2. Find the first node of $\mathtt{L}_f$ by computing $\alpha'_1 := \mathtt{T}_d[\tau_1] \oplus \tau_2$.
  3. For $1 \leq i \leq \#\bar{f}$:
     (a) Decrypt $\mathtt{D}_i$ by computing $\langle \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \mu \rangle := \tilde{\mathtt{D}}_i \oplus H_2(\tau_3, r)$, where $\mathtt{D}_i = (\tilde{\mathtt{D}}_i, r) := \mathtt{A}_d[\alpha'_i]$.
     (b) Delete and free $\mathtt{D}_i$ by setting $\mathtt{A}_d[\alpha'_i]$ to a random $(6 \cdot \mathsf{len}_{\mathsf{addr}} + |\mu| + \lambda)$-bit string.
     (c) Delete and free $\mathtt{D}_i$'s dual by setting $\mathtt{A}_s[\alpha_4]$ to a random $(\mathsf{len}_{\mathsf{id}} + \mathsf{len}_{\mathsf{addr}} + \lambda)$-bit string.
     (d) Let $\mathtt{N}_{-1}$ be the node that precedes $\mathtt{D}_i$'s dual. Update $\mathtt{N}_{-1}$'s "next pointer" by setting: $\mathtt{A}_s[\alpha_5] := (\beta_1 \oplus \langle 0^{\mathsf{len}_{\mathsf{id}}}, \alpha_4 \oplus \alpha_6 \rangle, r_{-1})$, where $(\beta_1, r_{-1}) := \mathtt{A}_s[\alpha_5]$. Also, update the pointers of $\mathtt{N}_{-1}$'s dual by setting
     $$\mathtt{A}_d[\alpha_2] := \big(\beta_2 \oplus \langle \mathbf{0}, \mathbf{0}, \alpha'_i \oplus \alpha_3, \mathbf{0}, \mathbf{0}, \alpha_4 \oplus \alpha_6, 0^{|\mu|} \rangle, r^*_{-1}\big),$$
     where $(\beta_2, r^*_{-1}) := \mathtt{A}_d[\alpha_2]$.
     (e) Let $\mathtt{N}_{+1}$ be the node that follows $\mathtt{D}_i$'s dual. Update $\mathtt{N}_{+1}$'s dual pointers by setting:
     $$\mathtt{A}_d[\alpha_3] := \big(\beta_3 \oplus \langle \mathbf{0}, \alpha'_i \oplus \alpha_2, \mathbf{0}, \mathbf{0}, \alpha_4 \oplus \alpha_5, \mathbf{0}, 0^{|\mu|} \rangle, r^*_{+1}\big),$$
     where $(\beta_3, r^*_{+1}) := \mathtt{A}_d[\alpha_3]$.
     (f) Set $\alpha'_{i+1} := \alpha_1$.
  4. Remove the ciphertext that corresponds to id from $\mathbf{c}$.
  5. Remove τ_1 from $\mathtt{T}_d$.

File decryption.

- Dec$\big(K, \{c_i\}_{i \in I}\big)$: For $i \in I$, let $f_i \leftarrow \mathsf{SKE.Dec}_{K_4}(c_i)$. Output $\{f_i\}_{i \in I}$.

Note here, we only give a mathematically self-contained description in this subsection, the detailed description of the operation flow in combination with other components will be deferred to Section 4.5, such as who will run the algorithm for each step. At the current stage, we expect to define the DSSE scheme at an abstract and make use of the construction as a black-box. This is mainly based on the following reasons:

1. Modules design ("plug & play") allows for greater flexibility of the overall architecture.
2. As we have mentioned in Section 3.4.2, all currently known SSE schemes still suffer some privacy issues. We expect that some more practical and secure schemes will be proposed in the near future, then we can easily replace the scheme without changing other components.
3. If some security vulnerabilities are found, we can easily make a drop-in replacement.

4.4 Flexible Attribute-Based Encryption Interface

Our DSSE scheme can be easily extended to support any ABE mechanisms that involve techniques of issuing keys that bound to the devices' attributes. These keys are called attribute keys. Only the requesting entity who has a matching attributes can decrypt the key and decrypt the contents in the key-policy ABE setting.

In the case of attestation raw data, the contents are stored actually in the ASSURED data storage engine. The index-based searchable encryption scheme is established between the attestation keywords and the corresponding location pointers (i.e., the `_uid`, see Chapter 5) to the ASSURED data storage engine where the attestation raw data are stored. And thus, the encryption of the contents stored on the ASSURED data storage engine are completely independent of the SSE scheme.

The contents stored on the ASSURED data storage engine can be encrypted using an independent ABE scheme. When a requesting entity launches a search request, and gets back a set of location pointers, the SCB retrieves the encrypted contents stored in the corresponding location pointers from the ASSURED data storage engine. In this case, any requesting entity will be able to have access to the (encrypted) raw data but only those that can produce the necessary attributes should be able to decrypt the encrypted data.

We make use of an ABE instantiation $\mathsf{ABE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ as a black-box and do not rely on its concrete construction. In other words, the dynamic searchable encryption scheme deployed in ASSURED can be combined with any ABE scheme. For an instantiation of the ABE scheme in ASSURED, we refer to Deliverables 4.1 [12] and 4.2 [17] where they describe an ASSURED-designed ABE scheme that allows ASSURED block chain users to perform ABE using their embedded TPMs. The proposed ABE will provide a secure attribute based encryption/decryption for the formatted data via fast and efficient encryption technology where attribute keys are stored into the TPM for extra protection and secure management of various encryption/decryption keys.

4.5 DSSE Secure Data Operation

Basically, we have two kinds of data to deal with for off- and on-chain data storage and access, one is the attestation raw data and the other is the attestation result data. In addition, as identified in ASSURED DLT Network, we employ a hybrid approach where we do not store full copies of the relevant attestation data on the blockchain for performance and efficiency consideration as well as scalability and sharing facilitation reasons. Instead, we make use of both the ledgers and off-chain storage to record the data; attestation result data is stored on the private ledger whereas the accompanying raw data (e.g., system traces, control-flow traces or digest lists of loaded binaries) is stored in the ASSURED Data Storage Engine.

Regarding the private ledger, the core storing data is

- the attestation identifier, e.g., attestation ID $\mathtt{attID}$,
- the corresponding attestation result,
- the (signed) location pointer to the corresponding encrypted attestation raw data where it is stored on the ASSURED Data Storage Engine, and

- the hash of the concatenation of the following data: the attestation identifier, the corresponding encrypted attestation raw data, and the location pointer to the corresponding encrypted attestation raw data stored in the ASSURED Data Storage Engine.

Note that, by the signed location pointer, we mean that the pointer is digitally signed by the SCB. And thus, data storage engine is able to verify if the given pointer is validated and granted by the SCB via the verification of the signature. And further, the hash value above is used to maintain the integrity of the storage data, linking with attestation ID and storage location.

On the public ledger, the information we store are

- a copy of the hash value stored in the private ledger (see the fourth item in the private ledger above), and
- DSSE related data which mainly includes the index-based database and the encrypted location pointers (associated with the copy of hash value corresponding to the same attID) to the ASSURED Data Storage Engine where attestation raw data is stored, allowing to query over those data efficiently.

We note that the encrypted pointers stored on the public ledger are not signed by the SCB before storage. They, instead, will be signed after decryption and before being sent to external parties. In this way, the signature can guarantee that the pointers held by the external parties are those granted by the SCB.

Blockchain Peers: Blockchain peers are considered as the main entities, along with the SCB, to perform the DSSE operations, and they are the ones who maintain the ledgers. We denote the Peers that performs operations on public and private ledgers as PeerA and PeerB, respectively. We note that these are only notations used to distinguish operations on public and private ledgers, and PeerA/B could be also a set of peers (note this depends how many peers we set per blockchain channel in the implementation stage).

Pointer: We remark that: the pointer mentioned above is not necessarily a specific real storage address pointer to the (encrypted) attestation raw data stored on the data storage engine, it can be a pseudo-pointer so that when this pseudo-pointer is given to external parties when the searches are done, the specific storage location will not be disclosed in particular. For example, the pseudo-pointer can be a pseudorandom permutation or an encryption of the real location pointer, we will discuss the related data storage structure in more details in Chapter 5. For simplicity, we still call it the location pointer if there is no ambiguity.

Workflow Overview: The full workflow of the designed DSSE on public ledger and data storage and access on private ledger are described as follows (see Figure 4.1 and 4.2). At a high level, the data to be uploaded is first transmitted from a PeerB to the SCB, then the SCB sends the encrypted attestation raw data to the data storage engine and gets the corresponding location (pseudo-) pointers. Next, the SCB computes the corresponding hash values, and sends the attestation bundle (hash values, attestation results, attestation ID, pseudo-pointers) back to the PeerB, so that the peer is able to merge those information on the private ledger. The SCB further generates the index-based structure related to the searchable function, and sends the index-based structure, encrypted location pointers and the keyword dictionary to the PeerA.

When it comes to searching, the external party first scans the public ledger, finds the interesting keywords and sends them to the SCB. The SCB then generates the search tokens, and request PeerA to perform the search function on the public ledger, retrieves and decrypts the encrypted location pointers. The pointers, with the SCB's digital signature, are then returned to the external

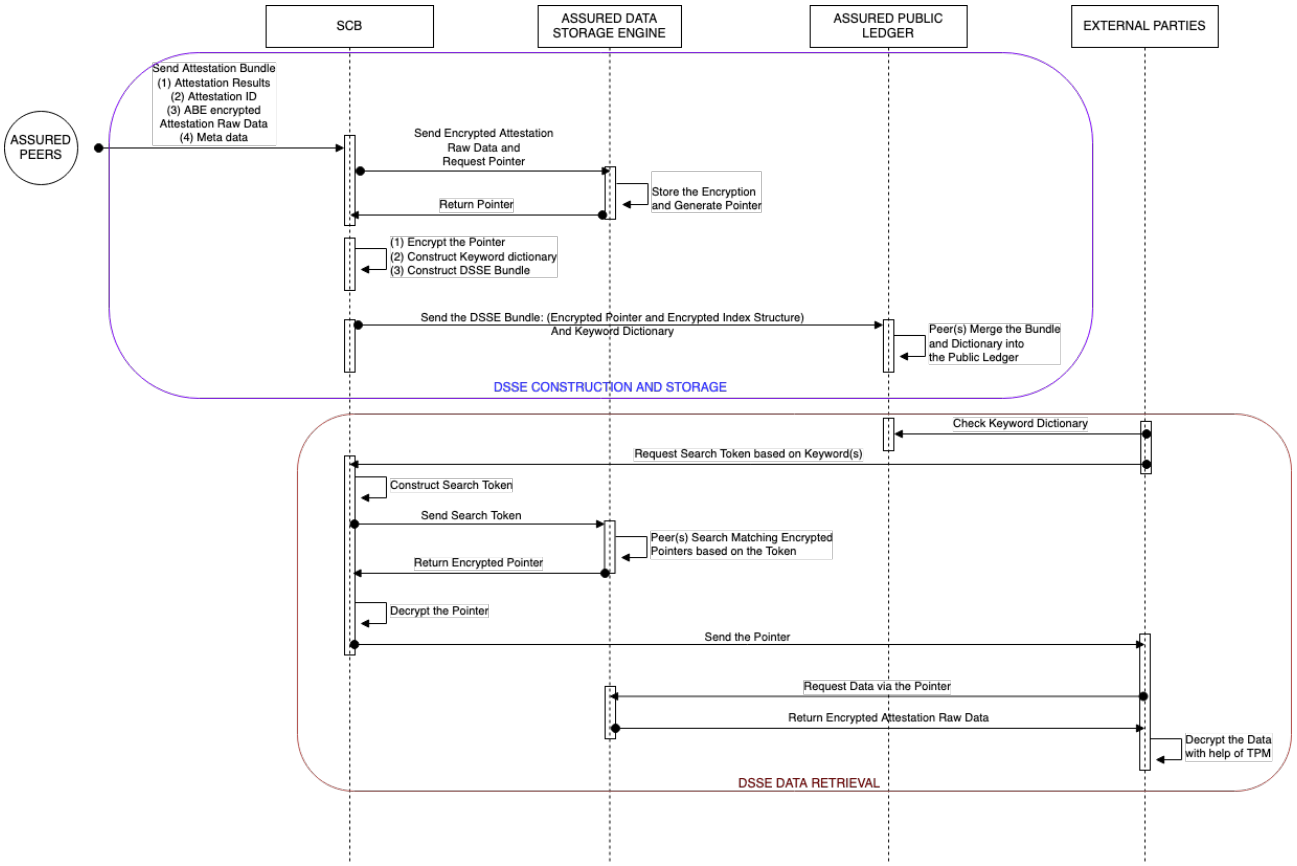

Figure 4.1: General Workflow of DSSE Secure Data Storage and Search on Public Ledger

party which will request the data storage engine to retrieve the relevant data. The data updates (including addition, deletion and updating an existing file) are almost identical to the data upload workflow.

Note we will present specific details on private ledger data access in Section 4.6. In the next subsections, we will give a step-by-step description about the operation flow of the dynamic searchable symmetric encryption in ASSURED framework.

Captured Security Requirements: DLT-SCT-03, DLT-SCT-05, DLT-SCT-07, DLT-SCT-30, DLT-SCT-52, DLT-SCT-55, DLT-DEC-06.

4.5.1 Updates on the Public Ledger

Due to tamper proof feature and the intrinsic design of the blockchain, direct update on previous settled blocks on the ledger is unlikely possible. One cannot directly modify the data stored on the ledger just as a real-world database does. However, we can keep putting updated contents as a new copy and then put this new copy into the new block. Finally, we make a statement/declaration (or a similar approach) that the old copy on the ledger is invalid. In this section, every time we refer to updating the relevant data on the ledger, we mean updating the data in the above approach.

Captured Security Requirements: DLT-SCT-50, DLT-SCT-54.

4.5.2 Building the Index Structure

Now we consider how to upload data for the first time and build the data related to the searchable function. The main flow of operations is as follows.

1. The data to be uploaded is transmitted from a PeerB to the SCB. The data mainly includes the attestation identifier (attID), the corresponding attestation result data (resultData), and the corresponding encrypted attestation raw data using an ABE scheme (ABE.Enc(rawData)) (see Section 4.4).

2. The SCB sends the encrypted attestation raw data ABE.Enc(rawData) to the ASSURED data storage engine which stores them on the engine. For each attestation identifier attID, the SCB gets the corresponding location pointer (ptr) from the storage engine where the encrypted attestation raw results are stored.

3. The SCB calculates, for each attestation identifier attID, the hash hashV of the concatenation of the following data: the attestation identifier attID, the corresponding encrypted attestation raw data ABE.Enc(rawData), and the location pointer ptr to the corresponding encrypted attestation raw data stored in the ASSURED Data Storage Engine.

4. The SCB sends the following data (quadruples) to the PeerB: the attestation identifier attID, the corresponding attestation result data resultData, the (signed by the SCB) location pointer ptr to the corresponding encrypted attestation raw data stored in the ASSURED Data Storage Engine, and the hash value hashV.

5. The PeerB writes the received data to the private ledger.

6. The SCB organizes the attestation result data (associated with their location pointers ptr's) according to some certain categories that will serve as search keywords, such as by device ID, date and time of the event, type of event, success or failure of event, etc. The SCB runs the **Setup** phase of the DSSE scheme described in Section 4.3.2 to set up the keywords universe W, public parameters and corresponding keys $K \leftarrow \mathsf{DSSE.Gen}(1^\lambda)$. The SCB treats the location pointer (ptr) as the file f, and establishes an inverted index-based relationship as described in Section 4.3 between the keywords and ptr's. **Note that here the keywords universe should include the keywords that may appear in the future.** For example, in the implementation stage, we may use the whole English dictionary to form the keywords universe, so that each English word could be used as a search keyword, e.g., Monday, Europe, etc.

7. The SCB runs the **Building the database** phase (DSSE.Enc) of the DSSE scheme described in Section 4.3.2 to obtain the encrypted index and a sequence of ciphertexts $(\gamma, \mathbf{c}) \leftarrow \mathsf{DSSE.Enc}(K, \overrightarrow{\mathsf{ptr}})$.

8. The SCB sends the encrypted index γ, the sequence of ciphertext-hash pairs (encrypted location pointer and the hash value corresponding to the same attID) (c_i, hashV_i)'s to the PeerA, and the PeerA writes them to the public ledger.

9. The SCB sends the keywords dictionary DictW to the PeerA. The PeerA writes them to the public ledger.

Captured Security Requirements: DLT-SCT-03, DLT-SCT-05, DLT-SCT-08, DLT-SCT-13, DLT-SCT-14, DLT-SCT-17, DLT-SCT-30, DLT-SCT-34, DLT-SCT-35, DLT-SCT-36, DLT-SCT-38, DLT-SCT-39, DLT-SCT-40/DLT-SCT-41, DLT-SCT-45, DLT-SCT-46, DLT-SCT-55, DLT-DEC-06, DLT-SHA-14.

4.5.3 Keyword (Fuzzy) Search

When a search query is needed, the process is initiated by an external party (EP) who wants to perform the search function. Here we only consider the search via a keyword on the public ledger, which we call fuzzy search. The search via a concrete attestation identifier, called concrete search, will be discussed in more details in Section 4.6. There, we will also consider the search requests from an internal party. The main flow of operations is described as follows.

1. The EP initiates an access request to the public ledger to the administrator of the public ledger.
2. The administrator of the public ledger enforces an ABAC to authenticate whether the EP has the matching attributes. If the EP has the matching attributes, go to Step 3. If not, abort and return $\perp$ (reject).
3. The EP requests the PeerA to access the keywords dictionary DictW on the public ledger and retrieves the interesting keywords.
4. The EP sends the keywords to the SCB.
5. For each keyword w sent by the EP, the SCB runs the search token generation algorithm ($\mathsf{DSSE.SrchToken}$) of the **Keyword search** phase in the DSSE scheme described in Section 4.3.2 to generate a search token $\tau_s \leftarrow \mathsf{DSSE.SrchToken}(K, w)$.
6. The SCB sends the search tokens τ_s's to the PeerA.
7. For each search token τ_s sent by the SCB, the PeerA performs the keyword search algorithm $\mathsf{DSSE.Search}$ on the public ledger to obtain a sequence of ciphertexts (encrypted location pointers) $\{c_i\}_{i\in I_w} \leftarrow \mathsf{DSSE.Search}(\gamma, \tau_s)$, and sends them back to the SCB.
8. The SCB computes the intersection or union $\{c_i\}_{i\in I}$ of all returned ciphertext sets (depending on the choice of EP).
9. The SCB performs the **File decryption** phase ($\mathsf{DSSE.Dec}$) of the DSSE scheme described in Section 4.3.2 to output the decrypted files $\{\mathsf{ptr}_i\}_{i\in I} \leftarrow \mathsf{DSSE.Dec}(K, \{c_i\}_{i\in I})$, signs and sends them back to the EP.
10. The EP sends the (signed) location pointers $\{\mathsf{ptr}_i\}_{i\in I}$ to the administrator of the ASSURED Data Storage Engine and requests retrieval of the (encrypted) attestation raw data.
11. The administrator of the ASSURED Data Storage Engine verifies whether the location pointers are signed by the SCB. If true, the administrator (maybe need to convert the pseudo-pointers to the corresponding real location pointers on the data store engine first) returns the (encrypted) attestation raw data $\{\mathsf{ABE.Enc}(\mathsf{rawData})\}_{\mathsf{ptr}_i}$ stored at the location (pseudo-) pointers $\{\mathsf{ptr}_i\}_{i\in I}$ in the ASSURED Data Storage Engine. If not, return $\perp$ (reject).

Note that the attestation raw data stored at the corresponding location pointers in the ASSURED Data Storage Engine is encrypted using an ABE scheme. Only the requesting entity who has the matching attributes can decrypt the key (with help of its TPM) and further decrypt the contents in the key-policy ABE setting.

Captured Security Requirements: DLT-SCT-03, DLT-SCT-05, DLT-SCT-08, DLT-SCT-21, DLT-SCT-22, DLT-SCT-24, DLT-SCT-25, DLT-SCT-30, DLT-SCT-34, DLT-SCT-35, DLT-SCT-36, DLT-SCT-38, DLT-SCT-39, DLT-SCT-40/DLT-SCT-41, DLT-SCT-45, DLT-SCT-46, DLT-SCT-47, DLT-SCT-55, DLT-DEC-06, DLT-SHA-14.

4.5.4 File Addition

We proceed to describe the operation flow of adding a new attestation raw data file. If multiple files are needed to be added, we will run the file addition flow for each file per round. The process is initiated by a data subject who creates and collects the data and chooses to perform the data addition function. The main flow of operations is quite similar to that of building the database.

1. The data to be added is transmitted from a PeerB to the SCB. The data mainly includes the attestation identifier (attID), the corresponding attestation result data ($\mathsf{resultData}$), and the corresponding encrypted attestation raw data using an ABE scheme ($\mathsf{ABE.Enc}(\mathsf{rawData})$).

2. The SCB sends the encrypted attestation raw data $\mathsf{ABE.Enc}(\mathsf{rawData})$ to the ASSURED data storage engine. The SCB obtains the corresponding pointer (ptr) from the engine where the encrypted attestation raw data are stored.

3. The SCB calculates the hash value hashV of the concatenation of the following data: the attestation identifier attID, the corresponding encrypted attestation raw data $\mathsf{ABE.Enc}(\mathsf{rawData})$, and the pointer ptr to the corresponding encrypted attestation raw data stored in the engine.

4. The SCB sends the following data (quadruple) to the PeerB: the attestation identifier attID, the corresponding attestation result data $\mathsf{resultData}$, the (signed by the SCB) pointer ptr, and the hash value hashV. We note that the PeerB has already known the attestation results and the attestation ID. Without loss of generality, we say that the above quadruple is sent to the peer.

5. The PeerB writes the received data (i.e. the quadruple) to the private ledger.

6. The SCB organizes the attestation result data (associated with its location pointer ptr) according to the keywords universe W. The SCB runs the token addition algorithm $\mathsf{DSSE.AddToken}$ of the **File addition** phase in the DSSE scheme described in Section 4.3.2 to generate an addition token and the encrypted location pointer $(\phi_a, c_{\mathsf{ptr}}) \leftarrow \mathsf{DSSE.AddToken}(K, \mathsf{ptr})$. The SCB then sends $\tau_a := (\phi_a, (c_{\mathsf{ptr}}, \mathsf{hashV}))$ to the PeerA.

7. The PeerA runs the **File addition** algorithm ($\mathsf{DSSE.Add}$) to update the index and encrypted location pointer (along with the associated hash value) on the public ledger, i.e. running $(\gamma', \mathbf{c}') \leftarrow \mathsf{DSSE.Add}(\gamma, \mathbf{c}, \tau_a)$.

File update. When updating an existing file, we can treat this case as adding a completely new file.

Captured Security Requirements: DLT-SCT-03, DLT-SCT-05, DLT-SCT-08, DLT-SCT-13, DLT-SCT-14, DLT-SCT-17, DLT-SCT-30, DLT-SCT-34, DLT-SCT-35, DLT-SCT-36, DLT-SCT-38, DLT-SCT-39, DLT-SCT-40/DLT-SCT-41, DLT-SCT-45, DLT-SCT-46, DLT-SCT-55, DLT-DEC-06, DLT-SHA-14.

4.5.5 File Deletion

Finally, we describe the flow of deleting a stored file. If multiple files are requested to be deleted, we will similarly run the file deletion for each file per time. The process is initiated by a data subject who owns the data and wants to perform the data deletion function. The main flow of operations is illustrated as follows.

1. The data deletion request is sent by a Peer (either PeerA or PeerB) to the SCB. This request mainly includes: attID, the optional (resultData), and the optional ABE.Enc(rawData). We note that deletion so far is set as a precise deletion (i.e. cannot support fuzzy deletion). In this case, the unique identifier, attID, is the crucial element we require; while other items could be optional.

2. With the attID, the PeerB can easily retrieve the corresponding back to the SCB.

3. Given ptr, the SCB runs the algorithm DSSE.DelToken of the **File deletion** phase in the DSSE scheme described in Section 4.3.2 to generate the deletion tokens and file identifier $(\tau_1, \tau_2, \tau_3, \mathsf{id}(\mathsf{ptr})) \leftarrow \mathsf{DSSE.DelToken}(K, \mathsf{ptr})$. The SCB then sends $\tau_d = (\tau_1, \tau_2, \tau_3, \mathsf{id}(\mathsf{ptr}))$ to the PeerA.

4. The PeerA runs the **File deletion** algorithm (DSSE.Del) to "delete" the index and encrypted location pointers (along with the associated hash values) on the public ledger $(\gamma', \mathbf{c}') \leftarrow \mathsf{DSSE.Del}(\gamma, \mathbf{c}, \tau_d)$. We state that here we enable the peer to run the deletion algorithm to reform the index structure and the encrypted pointers in the algorithm level. However, due to the tamper-proof feature of the ledger, the previously stored data on the ledger is impossible to easily deleted. Thus, after running the deletion algorithm, the PeerA must announce an additional piece of data on the ledger so as to declare a "null" (or invalid) value to the corresponding data block, e.g., the transaction ith (with an encrypted pointer) in jth is "null", and this piece of data could be wrapped as a new transaction to be stored on a new block.

5. The SCB also requests the data storage engine to delete the encrypted attestation raw data stored at the location (pseudo-) pointer ptr as well as terminating the use of the pointer.

6. As for the private ledger, the SCB requests the PeerB to make a similar statement/declaration that the relative item (the pointer in a specific transaction) corresponding to the attestation identifier attID is invalid or null.

Captured Security Requirements: DLT-SCT-03, DLT-SCT-05, DLT-SCT-08, DLT-SCT-13, DLT-SCT-14, DLT-SCT-17, DLT-SCT-30, DLT-SCT-34, DLT-SCT-35, DLT-SCT-36, DLT-SCT-38, DLT-SCT-39, DLT-SCT-40/DLT-SCT-41, DLT-SCT-45, DLT-SCT-46, DLT-SCT-55, DLT-DEC-06, DLT-SHA-14.

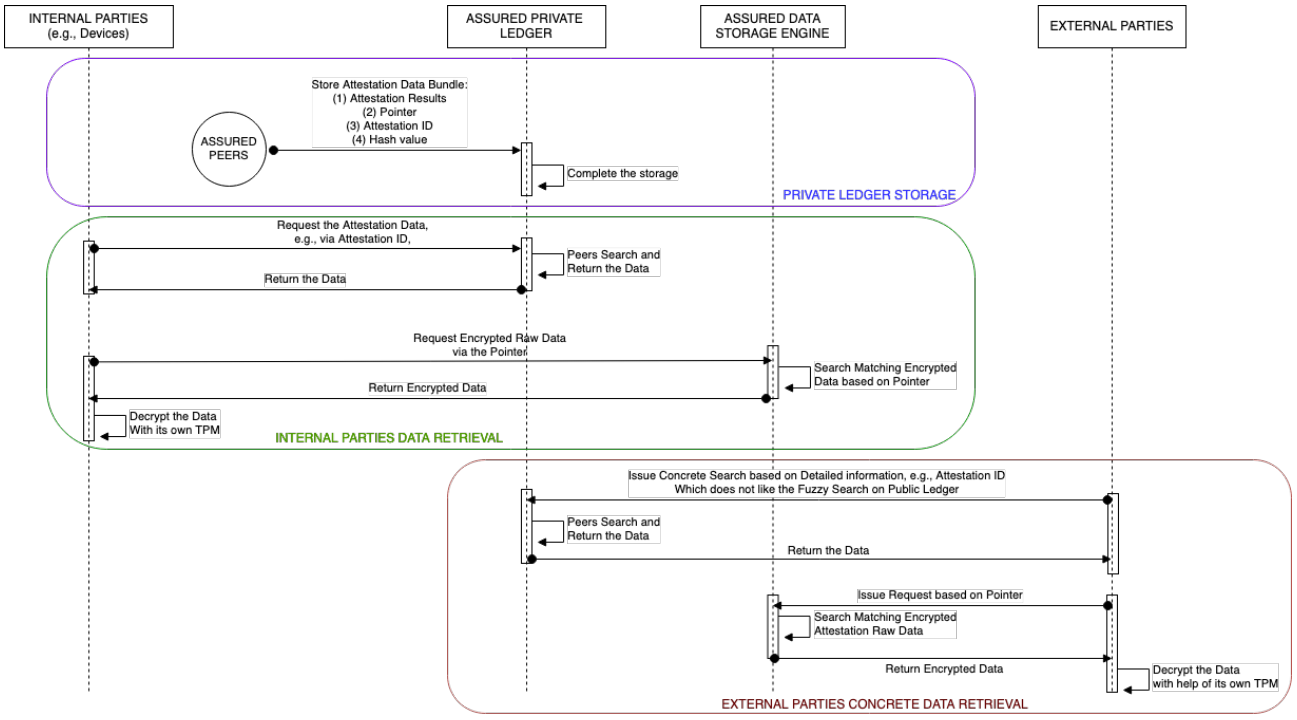

Figure 4.2: General Workflow of Secure Data Storage and Search over the Private Ledger

4.6 Private Ledger Data Access

We have described in Section 4.5.3 the searches via keywords on the public ledger, the so-called fuzzy search. Recall that the attestation identifiers and (signed) location pointers are also stored on the private ledger without encrypting, which means if the requesting party has the permission to directly access the private ledger, the requesting party can directly perform a concrete search on the private ledger and obtain the interesting location pointers without having to run the SSE protocol. This search over the private ledger is different from that of the fuzzy search (via keywords) over the public ledger. Thus, in this section, we will describe the concrete searches on the private ledger in case the requesting party has the permission to directly access the private ledger.

Here we also consider the internal party such as the device who submits/outsources the data and (of course) has the the permission to directly access the private ledger. The access to the private ledger data is depicted in Figure 4.2. We summarize it as follows.

Store data on the private ledger. Firstly, recall that the data is submitted to a PeerB by the internal parties (e.g. devices). The data is a tuple of (attID, resultData, and ABE.Enc(rawData)). We note that this is different from the case of public ledger storage. In this case, the data could directly come from the internal parties, devices.

Secondly, similar to the case of private ledger data storage, the SCB gets the (pseudo-) pointer (ptr) from the data storage engine. Then, the SCB calculates, for each attestation identifier attID, the hash hashV of the concatenation of the following data: the attestation identifier attID, the corresponding encrypted attestation raw data ABE.Enc(rawData), and the location pointer ptr to the corresponding encrypted attestation raw data stored in the ASSURED Data Storage Engine.

Thirdly, the SCB sends the following data (quadruples) to the PeerB: the attestation identifier attID, the corresponding attestation result data resultData, the (signed by the SCB) location

pointer ptr to the corresponding encrypted attestation raw data stored in the ASSURED Data Storage Engine, and the hash value hashV.

Finally, the PeerB writes the received data to the private ledger. Concretely, the information stored on the private ledger is a list of quadruples: $\{\mathsf{attID}, \mathsf{resultData}, \mathsf{ptr}, \mathsf{hashV}\}_i$.

Search request by an internal party. As the internal party (e.g., devices) submitting the data, it knows the exact attestation identifier with the access permission over the private ledger. This internal party can send the attestation identifiers to the PeerB, so that the peer searches the data on the ledger and returns the related data (including the plaintext pointers). Then the internal party sends the (pseudo-) pointers to the ASSURED data storage engine and requests to retrieve data. The data storage engine verifies the signatures and returns the encrypted attestation raw data $\mathsf{ABE.Enc}(\mathsf{rawData})$ if passed. Finally, the internal party decrypts the returned data using its own TPM and obtains the original attestation raw data, because the data is actually encrypted and submitted by this internal party.

Search request by an external party. Similarly, when the external party knows the exact attestation identifier, then he/she initiates an access request to the private ledger to the administrator of the private ledger via ABAC. If passed, the follow-up workflow will be almost the same as that of internal party except for the last step that only if the external party has the matching attributes he/she can decrypt the key and thus decrypt the contents in the key-policy ABE setting.

Captured Security Requirements: DLT-SCT-03, DLT-SCT-05, DLT-SCT-08, DLT-SCT-21, DLT-SCT-22, DLT-SCT-24, DLT-SCT-25, DLT-SCT-34, DLT-SCT-35, DLT-SCT-40/DLT-SCT-41, DLT-SCT-55, DLT-SHA-14.

4.7 Access via the ABAC

In the ASSURED blockchain network, we use ABAC mechanism to maintain access control. This control is supported by the blockchain peers (with given users' credentials) and the SCB (with help of smart contracts based on access policies). We say that the access is requested by both internal and external parties, e.g., devices and external auditor; and the access to the private and public ledgers are considered. Recall that any parties which would like to register to any ledger should send request to the SCB, so that the SCB can use the access policy based smart contract to check if the parties are permitted to access to the ledger based on their attributes. If the check is valid, the SCB will refer them to the blockchain CA. They will be granted credentials by the blockchain CA. On a credential, we define the access attributes which are granted to a blockchain user. With this credential, this user is allowed to access to the blockchain network. And further this credential decides if this user is able to enter the public or the private channel. As for the internal parties, they are usually granted the credentials that enable them to join the private channel, in this case, they can easily perform read and write requests to the peers on the private ledger. If the credentials the users have do not match the corresponding access rights, the peers can easily turn down the requests. As for the external parties, they are initially given the access to the public channel. And similarly, the public ledger peers can handle the parties' requests on the ledger via their credentials. The parties can also execute some fuzzy search, via the DSSE, on the public ledger. Specifically, an external party, after knowing the keyword dictionary from the public ledger, issues one or some keywords to the SCB. The SCB will first check if this party is allowed to access to the attestation raw data. This can be done by calling the smart contract method for attribute verification. If the verification is valid, then the SCB will proceed to search tokens generation, and later, decrypt the pointers to the party. Here, the SCB

will make a digital signature on the pointers, so that the data storage engine can be notified that this is a valid search pointers issued by the SCB. Any issuing pointers without the SCB signature, the data storage engine will ignore. In addition to the fuzzy search, the external party is also allowed to issue a concrete search directly over the private ledger. To this end, the party should directly send the concrete request to the SCB. The SCB will run the smart contract to verify its attributes. If valid, the party will be handled by the blockchain CA to add it into the private channel. In this way, the party can exactly do search queries like the internal parties.

Captured Security Requirements: DLT-SCT-21, DLT-SCT-22, DLT-SCT-24, DLT-SCT-25.

Chapter 5

ASSURED Data Storage Engine

As defined in the ASSURED DLT architecture, the overall deployment of the ASSURED ecosystem includes a secure data storage facility that is used in collaboration with the ASSURED DLT ledger to store data that is essential for the overall operation of the concept. This engine is deployed and managed by an organisation that is operating ASSURED and as depicted in the high level architecture presented in Deliverable D4.1 [12]. There is no common cloud-based storage engine to facilitate all ASSURED deployments, but rather each organisation is responsible for its own data lake.

This chapter provides the details about which data will become available over the Data Storage Engine, and about the services the Engine will provide as well as the deployment options of the Engine.

5.1 Data Storage Engine and relevance to ASSURED Operations

The overall Data Storage Engine that will be used in ASSURED consists of two major parts; the database system that is used for **storing the attestation raw data** (already encrypted using ABE), and an indexing service that will be used in conjunction with the database to allow for faster access and querying over the encrypted data stored in the database.

As identified in the overall ASSURED blockchain architecture, the main scope of the Data Storage Engine is to hold data relevant to attestation raw data in an encrypted manner, so that these data are not stored on the ledgers for both performance issues as well as for facilitating data exchange with other stakeholders. In this context, the database will not hold attestation raw data that can be read by stakeholders, but will rather include encrypted blobs of information that will be able to be located using pointers stored on the ledger, and be decrypted by the different entities using ABE decryption. In any case, access to this Data Storage Engine will be only provided through the SCB which will be the component that will be responsible to write and read data to/from the Data Storage Engine. It is the component that will store the attestation results on the database (and provide back a pointer to each ledger), and also will be able to interpret the different pointers coming from the ledgers (either encrypted or decrypted) and retrieve the according information from the database.

As the main information to be stored on this database is that of attestation reports, the selection

for the database technology is that of a noSQL[1] and in this direction MongoDB[2] has been selected, however, any other similar database technology can be selected, as this is a facility that can operate as plug & play in the sense that no complicated or special database operations are to be requested from the storage engine. The main information to be stored in the database, includes documents that have a very simple format which can be all stored under a Single Collection and those are of the following format:

```
{
    _uid: "5099803df3f4948bd2f98391",
    update_record_timestamp: (Timestamp ""),
    entry_record_timestamp: "20211231235959",,
    attestation_timestamp: "20211231235858",
    encrypted_attestation_raw_data: "chiphertext"
}
```

• The `_uid` is an auto-generated identifier for each document, which will act as the pointer which will be returned to the SCB (alongside with the actual location of the database `databasepath` where this is stored (in order to facilitate the distributed deployment options as described in Section 5.3) to note the exact location in the database of the ABE-encrypted attestation raw data.

• The `update_record_timestamp` is the auto-generated timestamp of the update/entry of the document in the database which is the exact time where an entry has been included/updated in the database.

• The `entry_record_timestamp` is the timestamp of the entry of the document in the database as taken out of the initial call made by the SCB to the database. This indicated the first time a document has been written into the database and is not updated if the document is updated (unlike the "`update_record_timestamp`" field).

• The `attestation_timestamp` is the timestamp relevant to when the attestation has been executed, and is forwarded by the SCB to the database, alongside with the encrypted attestation data blob (see below).

• The `encrypted_attestation_raw_data` is the encrypted blob of the whole attestation raw data, which has been already encrypted by a device using ABE based on the keys it posses in its TPM Wallet. This raw data file is generated at the device level, and after being encrypted is provided to the SCB which in turn forwards it to the Secure Data Storage Engine.

Regarding the indexing service, this will be based on the ElasticSearch[3] technology in order to be able to index the MongoDB towards allowing the SCB to get responses faster to the queries it will perform over the databases. As such, an index of the whole database will be generated and updated constantly, in order to be able to overcome certain read-access performance pain points that may exist due to the nature of the noSQL approach.

5.2 Securing the Data Storage Engine

The Data Storage Engine of the overall infrastructure does contain only encrypted data (attestation raw data), which can be decrypted only by entities that posses the correct attributes to

[1] https://www.mongodb.com/nosql-explained
[2] https://www.mongodb.com/
[3] https://www.elastic.co/what-is/elasticsearch

engage with the ABE scheme applied, however, two more steps have been taken in order to strengthen the security of the overall data repository.

Firstly, access to the Data Storage Engine, as indicated in deliverable D4.1 [12] is granted only to the SCB, who is the main responsible for writing, updating and reading documents from the database. As such, only the SCB and (and some other entities which are provided with specific access rights) will be allowed to access the database and all requests will be done through the SCB (e.g., in the way of requesting the pointers). Also as identified in deliverable D4.1 [12], the SCB will not serve external parties with the real pointers to the database (the "`_uid`" field identified in Section 5.1, but will instead auto-generate a pseudo-pointer (a 16-character string) which will be the one to be provided to the external stakeholders. This pseudo-pointer will be created at real-time within the SCB, and will be stored temporarily in a table holding the different pseudo-pointers and the real pointers (provided as a combination of "`databasepath`" and "`_uid`") to the exact location of each entry. This record will have a short "`t=time-to-leave`" and as such any pseudo-pointer generated by the SCB will become obsolete after a certain period of time "`t`" measured in seconds. And the data storage engine will be thus protected from rogue calls by malicious entities so that they have to guess the pseudo-pointers (and will be also put on a connection blacklist by the SCB in case they continuously request non-existent pseudo-pointers).

As such, the SCB will keep a list of these values, as shown in the next table.

timestamp	pseudo-pointer	timetoleave	databasepath	uid
20220212232358	`f18af9ae92eb11ec`	1200	dbhostA	`176BF052-4ECB-4E1D-B48F-F3523F7F277F`
20220212232359	`b90ew42acc12es00`	1200	dbhostB	`04054007-0203-0001-0F0E-0D0C0B0A0980`
20220212232458	`f18afd1492eb11a3`	1200	dbhostA	`00010203-0405-4007-8009-0A0B0C0D0E0F`

Secondly, the Data Storage Engine will be also continuously checked by the SCB relevant to its integrity in order to be able to guarantee that no changes have been performed by any other entity on the data it holds. In order to do this, a snapshot of the engine infrastructure will be taken and attested resulting in an integrity check hash, which will be checked by the SCB prior to any interaction in order to check whether the actual engine is in its correct operating state, or some unauthorised changes have been applied to it.

5.3 Storage Engine Deployment Options

The Storage Engine is envisaged to operate in parallel to the DLT network and support it by offering the possibility to various entities to utilise this storage facility rather than putting data on the ledger. The Storage Engine is conceptually designed as a single deployment that covers an organisation where ASSURED is deployed, as mentioned above. However, this does not necessarily mean that the Storage Engine is a single infrastructure within an organisation, as it could be that various deployments can be used within the same ASSURED ecosystem. In fact, it is the utilisation of pointers in the ledger and their process by the SCB that allows for deploying multiple instances of the same Storage Engine in different settings within a single organisation, to facilitate the operation of the overall ASSURED system. This can be done for scalability reasons, while different department may choose to utilise their own infrastructure rather than moving data to other locations, even if those are of the same stakeholder, improving also data resilience in this way. As such, the deployments can be performed at the level of departments, or at any other conceptual division of the operations to be supported by the ASSURED framework within an organisation.

Following this approach, distributed data operations are supported in ASSURED as well, to ensure the improved stability of the overall network and refrain from having single point of failures when it comes to data access between organisations, placing the SCB in the center of attention as it is an intelligence proxy that is able to discover and query the different engines, whether these are operating within its organisation boundaries, or they are deployed elsewhere, belonging to other parties.

Nevertheless, ASSURED can operate having either a centralised storage engine deployment (as initially defined in the architecture), or by having distributed deployments of the same Storage Engine instance (as described above), and it is up to the use cases to decide which deployment scenario is more efficient, performant and cost-effective for their operation needs, considering also their available infrastructures and resources.

Chapter 6

Conclusion

This section presents conclusions over the deliverable and summarizes the main findings. Following the instructions output by D4.1 [12] and D1.4 [13], the scope of the deliverable focus on the use of SE this cryptographic primitive to provide secure and privacy-preserving data search and sharing for external parties and investigate how to integrate with ABAC to perform data access for internal parties. And meanwhile, the integration with ABAC, SCB and data storage engine is introduced to capture secure data storage and access. D4.3 delivers the detailed for the first stage development on data storage and access, and further reflects the design on the security and privacy requirements defined in D1.4 [13].

In summary, deliverable D4.3 introduces the following core data sharing components:

- DSSE: To support the secure data search and sharing for external parties in the ASSURED blockchain framework, Deliverable D4.3 presents a secure and dynamic searchable mechanism. In this mechanism, the SCB is allowed to construct searchable index structure, and public the keyword dictionary on the public ledger, so that external parties can use the keywords to issue search request to the SCB. The SCB is able to construct a search token for the public ledger peer(s) to perform a secure data search and return the corresponding requested file(s). This first stage DSSE development captures update operations on public ledger, index structure construction, keyword search and file addition/deletion stages. The current design can surely guarantee that: (1) all data stored on the public ledger are encrypted so as to protect attestation data confidentiality; (2) even the public ledger peer(s) will not have any idea on what data the requesters are querying, in this case the search privacy is maintained; (3) hash value of data storage engine pieces is stored on the ledger, so as to protect the integrity of the off-chain storage copy. In the DSSE, the interaction among the SCB, peer(s), external parties and the data storage engine are described in detail.
- Data Storage Engine: This engine is used to store the off-chain ABE encrypted attestation raw data. The design of the engine must have a tight connection with the public ledger, which relies on the pointer. The first stage of the engine description is presented in this deliverable. How the connection with the public ledger is built and some related secure approaches may be used to protect storage location of those encrypted data are considered.
- Other Aspects: This deliverable considers how the internal parties can access those data stored on the private ledger. This mainly depends on the ABAC mechanism that may check the credentials of the internal parties and their correspond read and write rights.

Beyond the development of the above components, the detailed Hyperledger Fabric membership service provider mechanism supporting ABAC, and the optimization on the DSSE and detailed

implementation of the SE and its interactions of other components will be further considered in the following deliverable D4.4. The integration of the proposed and implemented mechanisms with other ASSURED technical parts will be considered in the WP5 and WP6.

List of Abbreviations

Abbreviation	Translation
AE	Authenticated Encryption
AES	Advanced Encryption Standard
ABAC	Attribute-Based Access Control
ABE	Attribute-Based Encryption
AK	Attestation Key
CA	Certification Authority
CBC	Cipher Block Chaining
CCA	Chosen-Ciphertext Attack
CFA	Control-Flow Attestation
CIV	Configuration Integrity Verification
CKA	Chosen Keyword Attack
CPA	Chosen-Plaintext Attack
CSR	Certificate Signing Request
CTR	Counter
DAA	Direct Anonymous Attestation
DLT	Distributed Ledger technology
DSSE	Dynamic Searchable Symmetric Encryption
EA	Enhanced Authorization
EK	Endorsement Key
EP	External Party
GSS	Ground Station Server
HMAC	Hash-based Message Authentication Code
IBE	Identity-Based Encryption
KGA	Keyword Guessing Attack
MAC	Message Authentication Code
MSPL	Medium-level Security Policy Language (MSPL)
NMS	Network Management System
Privacy CA	Privacy Certification Authority
Prv	Prover
PCR	Platform Configuration Register
PLC	Program Logic Controller
PRF	Pseudorandom Function
RA	Risk Assessment
RAT	Remote Attestation
SCB	Security Context Broker
SE	Searchable Encryption
SoS	Systems of Systems
SSE	Searchable Symmetric Encryption
SSR	Secure Server Router
S-ZTP	Secure Zero Touch provisioning
TC	Trusted Component
TLS	Transport Layer Security
TPM	Trusted Platform Module

Abbreviation	Translation
Vf	Virtual Function
VM	Virtual Machine
Vrf	Verifier
WP	Work Package
ZTP	Zero Touch Provisioning

References

[1] Ghous Amjad, Seny Kamara, and Tarik Moataz. Forward and backward private searchable encryption with SGX. In *Proceedings of the 12th European Workshop on Systems Security, EuroSec@EuroSys 2019, Dresden, Germany, March 25, 2019*, pages 4:1–4:6. ACM, 2019.

[2] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 1996.

[3] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004.

[4] Raphael Bost. $\sum o\varphi o\varsigma$: Forward secure searchable encryption. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1143–1154. ACM, 2016.

[5] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1465–1482. ACM, 2017.

[6] Jin Wook Byun, Hyun Suk Rhee, Hyun-A Park, and Dong Hoon Lee. Off-line keyword guessing attacks on recent keyword search schemes over encrypted data. In Willem Jonker and Milan Petkovic, editors, *Secure Data Management, Third VLDB Workshop, SDM 2006, Seoul, Korea, September 10-11, 2006, Proceedings*, volume 4165 of *Lecture Notes in Computer Science*, pages 75–83. Springer, 2006.

[7] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society, 2014.

[8] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean

queries. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 353–373. Springer, 2013.

[9] Javad Ghareh Chamani, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. New constructions for forward and backward private symmetric searchable encryption. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 1038–1055. ACM, 2018.

[10] Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In John Ioannidis, Angelos D. Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security, Third International Conference, ACNS 2005, New York, NY, USA, June 7-10, 2005, Proceedings*, volume 3531 of *Lecture Notes in Computer Science*, pages 442–455, 2005.

[11] Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 577–594. Springer, 2010.

[12] The ASSURED Consortium. Assured blockchain architecture. Deliverable D4.1, November 2021.

[13] The ASSURED Consortium. Assured blockchain architecture. Deliverable D1.4, November 2021.

[14] The ASSURED Consortium. Assured reference architecture. Deliverable D1.2, May 2021.

[15] The ASSURED Consortium. Assured use cases & security requirements. Deliverable D1.1, February 2021.

[16] The ASSURED Consortium. Policy modelling & cybersecurity, privacy and trust constraints. Deliverable D2.2, November 2021.

[17] The ASSURED Consortium. Assured secure distributed ledger maintenance & data management. Deliverable D4.2, February 2022.

[18] The ASSURED Consortium. Assured tc-based functionalities. Deliverable D4.4, February 2022.

[19] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006*, pages 79–88. ACM, 2006.

[20] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. *J. Comput. Secur.*, 19(5):895–934, 2011.

[21] Ioannis Demertzis, Javad Ghareh Chamani, Dimitrios Papadopoulos, and Charalampos Papamanthou. Dynamic searchable encryption with small client storage. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020.

[22] Ge Gao, Lei Wu, and Yunxue Yan. A secure storage scheme with key-updating in hybrid cloud. *Int. J. High Perform. Comput. Netw.*, 13(2):175–183, 2019.

[23] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. TWORAM: efficient oblivious RAM in two rounds with applications to searchable encryption. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 563–592. Springer, 2016.

[24] Eu-Jin Goh. Secure indexes. *IACR Cryptol. ePrint Arch.*, page 216, 2003.

[25] Kun He, Jing Chen, Qinxi Zhou, Ruiying Du, and Yang Xiang. Secure dynamic searchable symmetric encryption with constant client storage cost. *IEEE Trans. Inf. Forensics Secur.*, 16:1538–1549, 2021.

[26] Thang Hoang, Attila Altay Yavuz, and Jorge Guajardo. Practical and secure dynamic searchable encryption via oblivious access on distributed data structure. In Stephen Schwab, William K. Robertson, and Davide Balzarotti, editors, *Proceedings of the 32nd Annual Conference on Computer Security Applications, ACSAC 2016, Los Angeles, CA, USA, December 5-9, 2016*, pages 302–313. ACM, 2016.

[27] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*. The Internet Society, 2012.

[28] Seny Kamara and Tarik Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 94–124, 2017.

[29] Seny Kamara and Charalampos Papamanthou. Parallel and dynamic searchable symmetric encryption. In Ahmad-Reza Sadeghi, editor, *Financial Cryptography and Data Security - 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers*, volume 7859 of *Lecture Notes in Computer Science*, pages 258–274. Springer, 2013.

[30] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 965–976. ACM, 2012.

[31] Kee Sung Kim, Minkyu Kim, Dongsoo Lee, Je Hong Park, and Woo-Hwan Kim. Forward secure dynamic searchable symmetric encryption with efficient updates. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1449–1463. ACM, 2017.

[32] Kaoru Kurosawa and Yasuhiro Ohtaki. Uc-secure searchable symmetric encryption. In Angelos D. Keromytis, editor, *Financial Cryptography and Data Security - 16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers*, volume 7397 of *Lecture Notes in Computer Science*, pages 285–298. Springer, 2012.

[33] Kaoru Kurosawa and Yasuhiro Ohtaki. How to update documents verifiably in searchable symmetric encryption. In Michel Abdalla, Cristina Nita-Rotaru, and Ricardo Dahab, editors, *Cryptology and Network Security - 12th International Conference, CANS 2013, Paraty, Brazil, November 20-22. 2013. Proceedings*, volume 8257 of *Lecture Notes in Computer Science*, pages 309–328. Springer, 2013.

[34] Jin Li, Yanyu Huang, Yu Wei, Siyi Lv, Zheli Liu, Changyu Dong, and Wenjing Lou. Searchable symmetric encryption with forward search privacy. *IEEE Trans. Dependable Secur. Comput.*, 18(1):460–474, 2021.

[35] D Sivaganesan. A data driven trust mechanism based on blockchain in iot sensor networks for detection and mitigation of attacks. *Journal of trends in Computer Science and Smart technology (TCSST)*, 3(01):59–69, 2021.

[36] Dawn Xiaodong Song, David A. Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 14-17, 2000*, pages 44–55. IEEE Computer Society, 2000.

[37] Xiangfu Song, Changyu Dong, Dandan Yuan, Qiuliang Xu, and Minghao Zhao. Forward private searchable symmetric encryption with optimized I/O efficiency. *IEEE Trans. Dependable Secur. Comput.*, 17(5):912–927, 2020.

[38] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical dynamic searchable encryption with small leakage. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society, 2014.

[39] Shifeng Sun, Ron Steinfeld, Shangqi Lai, Xingliang Yuan, Amin Sakzad, Joseph K. Liu, Surya Nepal, and Dawu Gu. Practical non-interactive searchable encryption with forward and backward privacy. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021.

[40] Shifeng Sun, Xingliang Yuan, Joseph K. Liu, Ron Steinfeld, Amin Sakzad, Viet Vo, and Surya Nepal. Practical backward-secure searchable encryption from symmetric puncturable encryption. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 763–780. ACM, 2018.

[41] Peter van Liesdonk, Saeed Sedghi, Jeroen Doumen, Pieter H. Hartel, and Willem Jonker. Computationally efficient searchable symmetric encryption. In Willem Jonker and Milan Petkovic, editors, *Secure Data Management, 7th VLDB Workshop, SDM 2010, Singapore, September 17, 2010. Proceedings*, volume 6358 of *Lecture Notes in Computer Science*, pages 87–100. Springer, 2010.

[42] Jiyi Wu, Lingdi Ping, Xiaoping Ge, Ya Wang, and Jianqing Fu. Cloud storage as the infrastructure of cloud computing. In *2010 International Conference on Intelligent Computing and Cognitive Informatics*, pages 380–383. IEEE, 2010.

[43] Xuanxia Yao, Zhi Chen, and Ye Tian. A lightweight attribute-based encryption scheme for the internet of things. *Future Gener. Comput. Syst.*, 49:104–112, 2015.

[44] Wei-Chuen Yau, Swee-Huay Heng, and Bok-Min Goi. Off-line keyword guessing attacks on recent public key encryption with keyword search schemes. In Chunming Rong, Martin Gilje Jaatun, Frode Eika Sandnes, Laurence Tianruo Yang, and Jianhua Ma, editors, *Autonomic and Trusted Computing, 5th International Conference, ATC 2008, Oslo, Norway, June 23-25, 2008, Proceedings*, volume 5060 of *Lecture Notes in Computer Science*, pages 100–105. Springer, 2008.

[45] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 707–720. USENIX Association, 2016.

[46] Minghao Zhao, Han Jiang, Zhen Li, Qiuliang Xu, Hao Wang, and Shaojing Li. An efficient symmetric searchable encryption scheme for dynamic dataset in cloud computing paradigms. *Int. J. High Perform. Comput. Netw.*, 12(2):179–190, 2018.