ETSI GR PDL 018 V0.0.3 (2023-01)

Redactable Distribued Ledgers

***Disclaimer***

The present document has been produced and approved by the Permissioned Distributed Ledger ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.  
It does not necessarily represent the views of the entire ETSI membership.

**Group Report**

Reference

DGR/PDL-018\_Redactable\_DL

Keywords

PDL, privacy

***ETSI***

650 Route des Lucioles

F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B

Association à but non lucratif enregistrée à la

Sous-Préfecture de Grasse (06) N° w061004871

***Important notice***

The present document can be downloaded from:  
<http://www.etsi.org/standards-search>

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the prevailing version of an ETSI deliverable is the one made publicly available in PDF format at [www.etsi.org/deliver](http://www.etsi.org/deliver).

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at <https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx>

If you find errors in the present document, please send your comment to one of the following services:  
<https://portal.etsi.org/People/CommiteeSupportStaff.aspx>

***Notice of disclaimer & limitation of liability***

The information provided in the present deliverable is directed solely to professionals who have the appropriate degree of experience to understand and interpret its content in accordance with generally accepted engineering or

other professional standard and applicable regulations.

No recommendation as to products and services or vendors is made or should be implied.

No representation or warranty is made that this deliverable is technically accurate or sufficient or conforms to any law and/or governmental rule and/or regulation and further, no representation or warranty is made of merchantability or fitness for any particular purpose or against infringement of intellectual property rights.

In no event shall ETSI be held liable for loss of profits or any other incidental or consequential damages.

Any software contained in this deliverable is provided "AS IS" with no warranties, express or implied, including but not limited to, the warranties of merchantability, fitness for a particular purpose and non-infringement of intellectual property rights and ETSI shall not be held liable in any event for any damages whatsoever (including, without limitation, damages for loss of profits, business interruption, loss of information, or any other pecuniary loss) arising out of or related to the use of or inability to use the software.

***Copyright Notification***

Reproduction is only permitted for the purpose of standardization work undertaken within ETSI.  
The copyright and the foregoing restrictions extend to reproduction in all media.

© ETSI 2022.

All rights reserved.

Contents

Intellectual Property Rights 4

Foreword 4

Modal verbs terminology 4

1 Scope 5

2 References 5

2.1 Normative references 5

2.2 Informative references 5

3 Definition of terms, symbols and abbreviations 6

3.1 Terms 6

3.2 Symbols 6

3.3 Abbreviations 6

4 Introduction to Redactable Distributed Ledger 7

4.1 Introduction 7

4.2 Issues with Immutable Ledgers 8

4.3 Redaction Operations 8

4.3.1 Introduction 8

4.3.2 Redaction Operations for Blockchain 8

4.3.3 Redaction Operations for Block DAGs 9

4.3.4 Redaction Operations for Blockless DAGs 9

5 Use Cases for Redactable Distributed Ledgers 10

5.1 Introduction 10

5.2 Identity Management 10

5.3 Smart Contracts 10

5.4 Data Sharing 11

6 Examples of Redactable Distributed Ledgers 11

6.1 Introduction 11

6.2 TCH-based Redactable Blockchain 11

6.2.1 Trapdoor-Controlled Hash 11

6.2.2 Blockchain Redaction Process 11

6.2.3 Blockchain Redaction Management 12

6.3 Policy-based Redactable Blockchain 13

6.4 Redaction Using New Block Structures 14

6.5 Discussions 14

7 Conclusions and Next Steps 14

7.1 Introduction 14

7.2 Recommendations for Next Steps 14

History 16

# Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (<https://ipr.etsi.org/>).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

**DECT™**, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™**and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

# Foreword

This Group Report (GR) has been produced by ETSI Industry Specification Group (ISG) Permissioned Distributed Ledger (PDL).

# Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the [ETSI Drafting Rules](https://portal.etsi.org/Services/editHelp!/Howtostart/ETSIDraftingRules.aspx) (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

# 1 Scope

This document will describe the definition of redactable distributed ledgers, presents use cases where redactable distributed ledgers are useful, and assesses existing solutions of redactable distributed ledgers. This document also discusses potential standardization areas for enabling, managing, and using redactable distributed ledgers.

# 2 References

## 2.1 Normative references

Normative references are not applicable in the present document.

[1] ETSI GS PDL 011: "Permissioned Distributed Ledger (PDL); Specification of Requirements for Smart Contracts' architecture and security".

## 2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non‑specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] D. Zhang, J. Le, X. Lei, T. Xiang, and X. Liao, “Exploring the Redaction Mechanisms of Mutable Blockchains: A Comprehensive Survey,” *International Journal of Intelligent Systems*, vol. 36, no. 9, 2021, pp. 5051-5084.

[i.2] G. Ateniese, B. Magri, D. Venturi, and E. Andrade, “Redactable Blockchain – or – Rewriting History in Bitcoin and Friends,” 2017 IEEE European Symposium on Security and Privacy (EuroS&P), 2017, pp. 111-126, doi: 10.1109/EuroSP.2017.37.

[i.3] Y. Sompolinsky, S. Wyborski, and A. Zohar, “PHANTOM GHOSTDAG: A Scalable Generalization of Nakamoto Consensus,” In Proceedings of the 3rd ACM Conference on Advances in Financial Technologies (AFT '21). Association for Computing Machinery, September 2, 2021, New York, NY, USA, 57–70 (https://doi.org/10.1145/3479722.3480990)

[i.4] S. Popov, “The Tangle,” White Paper Version 1.4.3, April 30, 2018 (http://www.descryptions.com/Iota.pdf)

[i.5] M. Mehar, et al., “Understanding a revolutionary and flawed grand experiment in blockchain: the DAO attack,” Journal of Cases on Information Technology (JCIT), 21(1), pp. 19-32, November 26, 2017 (https://papers.ssrn.com/sol3/papers.cfm?abstract\_id=3014782).

[i.6] ETSI GR PDL 004: “Permissioned Distributed Ledger (PDL); Smart Contracts System Architecture and Functional Specification”.

[i.7] I. Puddu, A. Dmitrienko, and S. Capkun, “uChain: How to Forget without Hard Forks,” 2017 (<https://eprint.iacr.org/2017/106>)

[i.8] D. Deuber, B. Magri and S. A. K. Thyagarajan, “Redactable Blockchain in the Permissionless Setting,” 2019 IEEE Symposium on Security and Privacy (SP), 2019, pp. 124-138, doi: 10.1109/SP.2019.00039.

[i.9] K. D. Richard, “A Data Structure for Integrity Protection with Erasure Capability,” NIST Cybersecurity Whitepaper (2018).

# 3 Definition of terms, symbols and abbreviations

## 3.1 Terms

For the purposes of the present document, the following terms apply:

**Trapdoor-Controlled Hash:** A hashing scheme with two modes: 1) collision-free one-way hashing without using a trapdoor key to map an input message to a unique hashing value, which is the typical mode in traditional collision-free hashing schemes; and 2) using a trapdoor key to cause a hashing collision (i.e., to cause the same hashing value for two different input messages).

Note: Chameleon hash [i.2] is an example of trapdoor-controled hash schemes.

**Hashing Collision:** The scenario where two different input messages gets the same hashing value using the same hashing function.

**Redaction:** A property for supporting changes to one or multiple objects on distributed ledgers.

**Redactble Objects:** The objects on distributed ledgers with the redaction property.

**Redaction Operations:** The actions or operations to change reactable objects on distributed ledgers.

Note: to modify, to delete, and/or to insert one or multiple redactable objects on distributed ledgers.

**Redactable Distributed Ledgers:** Distributed ledgers whose content can be modified with consensus through certain redaction operations.

**Trapdoor Key:** A secret key that allows the owner of this secret key to generate a hashing collision for two different input messages.

## 3.2 Symbols

Void.

## 3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

CH Chameleon Hash

DAG Directed Acyclic Graph

DAO Decentralized Autonomous Organization

ETSI European Telecommunications Standards Institute

EU European Union

GDPR General Data Protection Regulation

PDL Permissioned Distributed Ledger

RDL Redactable Distributed Ledger

TCH Trapdoor-Controlled Hash

TXN Transaction

# 4 Introduction to Redactable Distributed Ledger

## 4.1 Introduction

**Editor’s Note:** This part could be reduced, especially if other sections 5 & 6 is not significantly longer than section 4 or if the details of those ledger structures can not be efficiently covered in sectoons 5 and 6. Check if there is any differences among those structures from redaction perspective.

Distributed Ledger Technology (DLT) systems have been evolving. For example, distributed ledgers can be formed in different structures as illustrated in **Figure 4.1-1**, such as blockchain (e.g., BitcoinTM, EthereumTM, Hyperledger FabricTM), block Directed Acyclic Graph (DAG) (e.g., PPHANTOM [i.3]), and blockless DAG (e.g., IoTA [i.4]). In general, DLT brings unique characteristics and advantages such as immutability, transparency, and decentralization.

* Blockchain: A linear topology with a set of chained blocks starting from the first genesis block. In other words, each block (except the genesis block) has one and only one parent block.
* Block DAG: Blocks are organized in a DAG. Usually, each block (except the genesis block) has more than one parent blocks.
* Blockless DAG: Transactions are directly organized in a DAG. Usually, each transaction (except the genesis transaction) has more than one parent transactions.



(a). Blockchain



(b). Block DAG



(c) Blockless DAG

**Figure 4.1-1: Structures of Distributed Ledgers**

DLT-based solutions are usually characterized by being immutable, tamper-proof, and decentralized, making them outperform centralized counterpart systems. Such unique characteristics of DLT fit perfectly with any decentralized applications, where trust is an issue. For example, blockchain guarantees the integrity and security of financial transactions in the economic sector by preventing double-spending frauds and protecting users’ assets from being tampered with.

## 4.2 Issues with Immutable Ledgers

However, these unique characteristics of DLT (especially immutability) could be misused and lead to some potential issues. **Figure 4.2-1** illustrates the potential issues and limitations of immutable ledger structure.

* First, some information published by normal users on the distributed ledgers may become sensitive and cause privacy concerns in the future, especially in public DLT systems; however, such privacy-concerned information cannot be removed from the distributed ledgers due to its immutability, which prevents “rights to be forgotten” as described in General Data Protection Regulation (GDPR).
* Similarly, misinformation could be added to the distributed ledgers by attackers and stay there forever.
* Third, crypto criminals and hackers could inject into distribute ledgers illegal contents forbidden by national or international laws.
* Decentralized Autonomous Organization (DAO) applications, the most significant smart contract applications in the EthereumTM platform, are another example of immutability misuse. Hackers and crypto criminals discovered logical flaws and vulnerabilities in the DAO smart contracts that led to transferring over $120.3 million worth of EthereumTM coins to their accounts [i.5], which could have been avoided if such flawed smart contracts had been modified. This problem was semi-cured by hard forking EthereumTM blockchain back in 2016 to delete the attackers’ transfer transactions.
* Finally, the immutability unavoidably causes a scalability issue in maintaining the append-only and ever-increasing distributed ledgers.



Figure 4.2-1: Potential Issues and Limitations from Immutable Distributed Ledger

## 4.3 Redaction Operations

### 4.3.1 Introduction

For different structures of distributed ledgers, redaction operations could be different. Even it is possible to conduct the same redaction operation for different distributed ledger structures, the complexity and implications of the redation operation could still be different.

### 4.3.2 Redaction Operations for Blockchain

Redaction operations for blockchain-like distributed ledgers could be:

* **Transaction-Level Redaction Operations:** These redaction operations aim to impose changes on one or multiple existing transactions being included in an existing block and add new transactions. The modification of these transactions within the same block will automatically change 1) the fingerprint (e.g., the Merkle tree root) of all transactions; 2) the content of this block and the hash value of this block. Transaction-level redaction operations include:
  + To modify an existing transaction, which could be any one or multiple fields of the existing transaction;
  + To remove an existing transaction from the corresponding block;
  + To insert a new transaction to an existing block;
  + To modify more than one existing transactions from the corresponding block;
  + To remove more than one existing transactions from the corresponding block; and
  + To insert multiple transactions to an existing block.
* **Single-Block-Level Redaction Operations:** These redaction operations are used to change an existing block or introduce a new block. The change of the existing block will change the hash value of the existing block. The introduction of new block needs to deal with how to maintain the hash-based blockchain structure. Blockchain-level redaction operations include:
  + To modify non-transaction-related fields of an existing block;
  + To remove an existing block; and
  + To insert a new block to the existing blockchain (not to append the new block to the existing blockchain).
* **Multiple-Blocks-Level Redaction Operations:** These redaction operations are used to introduce changes related to multiple blocks, such as:
  + To remove multiple existing consecutive blocks;
  + To remove multiple existing non-consecutive blocks, which will essentially be using “Single-Block-Level Redaction Operations”;
  + To insert multiple new blocks in a consecutive order to the existing blockchain (not to append blocks to existing blockchain); and
  + To insert multiple new blocks in a non-consecutive order to existing blockchain (not to append blocks to existing blockchain)

### 4.3.3 Redaction Operations for Block DAGs

Redaction operations for a block DAG are similar to redaction operation to a blockchain.

* **Transaction-Level Redaction Operations:** They are same as transaction-level redaction operations for a blockchain as descried in clause 4.3.2.
* **Single-Block-Level Redaction Operations:** These redaction operations are used to change an existing block or introduce a new block. Each node on a block DAG represents an existing block. Since an existing block in a block DAG may be a child node of multiple parent nodes and/or a parent node of multiple child nodes, to change an existing block or to insert a new block needs to consider any impact and implications to its child and/or parent nodes. Blockchain-level redaction operations for a block DAG include:
  + To modify non-transaction-related fields of an existing block;
  + To remove an existing block; and
  + To insert a new block to the existing block DAG (not to append the new block to the existing block DAG).
* **Multiple-Blocks-Level Redaction Operations:** Thes redaction operations are used to introduce changes related to multiple blocks, such as:
  + To remove multiple existing connected blocks;
  + To remove multiple existing non-connected blocks;
  + To insert multiple new blocks in a connected sub-graph to the existing block DAG (not to append blocks to existing block DAG); and
  + To insert multiple new blocks in a non-connnected way to existing block DAG (not to append blocks to existing block DAG)

### 4.3.4 Redaction Operations for Blockless DAGs

Redaction operations for blockless DAGs are transaction-level only, since transaction is the only object unit in a blockless DAG. Each node on a blockless DAG represents an existing transaction. Since an existing transaction in a blockless DAG may be a child node of multiple parent nodes and/or a parent node of multiple child nodes, to change an existing transaction or to insert a new transaction needs to consider any impact and implications to its child and/or parent nodes.

The following redaction operations are possible for a blockless DAG:

* To modify an existing transaction, which could be any one or multiple fields of the existing transaction;
* To remove an existing transaction from the current blockless DAG;
* To insert a new transaction to the current blockless DAG;
* To modify multiple connected transactions from the current blockless DAG;
* To remove multiple connected transactions from the corresponding block; and
* To insert multiple transactions to the current blockless DAG (not to append them as leaf nodes)

# 5 Use Cases for Redactable Distributed Ledgers

## 5.1 Introduction

Redactble distributed ledgers are useful for some PDL-based applications asuch as identity management, smart contracts, and data sharing.

## 5.2 Identity Management

When PDL is used to support distributed identity management, each user creates its own user-centric identifier and stores the identrifier on distributed ledgers. The user requests and obtains its credentials from a third-party issuer. Then the user presents the identifier and credentials to a verifier when the user needs to consume services from the verifier. When the verifier verifies the credentials, it does not need to get any extra information from the issuer, but some public information about the ways to verify and authenticate the credentials (e.g., the public key of the issuer) from distributed ledgers. Note that such credential public information has also been stored on distributed ledgers.

Redactable distributed ledgers can be used to: 1) revoke an existing identifier; 2) modify credential public information; 3) remove an identifier (if there is potential privacy leakage).

* **Revoke an Existing Identifier:** The existing identifier of a user has been stored on distributed ledgers. When this identifier needs to be revoked it can also be removed from distributed ledgers. Assume this identifier has been included in an existing transaction. To remove this identifier from distributed ledgers, redaction operations can be used to update this transaction by deleting the identifier from it or simply remove the entire transaction itself.
* **Modify Credential Public Information:** Credential public information describes how a corresponding credential should be verified. Since credential public information are stored on distributed ledgers, it can be accessed by any authorized entities especially verifiers. The way to verify and authenticate a credential could be changed; as a result, the credential public information on distributed ledgers needs to be modified accordingly. Redaction operations can be used for this purpose.
* **Remove an Identifier:** User identifiers are public information and also stored on distributed ledgers. Although it is hard to infer a user’s real identity from its user-centric identifier (e.g., a decentralized identifier), attackers could still be able to use techniques such as identity linkage to derive some sensitive information about the user and lead to potential privacy leakage. To prevent such privacy leakage, a user (or PDL governance service) can use redaction operations to remove an existing identifier from distributed ledgers.

## 5.3 Smart Contracts

Smart contracts as defined in ETSI GR PDL 004 [i.6] and ETSI GS PDL 011 [1] are auto-executalbe code or program, which stored on distributed ledgers. A user can create a smart contract to distributed ledgers and/or trigger to execute an existing smart contract. After a smart contract is deployed and stored on distributed ledgers, it can not be updated although an on-chain smart contract can be destroyed if it embeds a destroy function.

Redactable distributed ledgers can be used to change a smart contract, for instance, to fix a design flaw in an on-chain smart contract, or to upgrade an on-chain smart contract.

* **Smart Contract Flaw Fixing:** Although a smart contract needs to be carefully designed and thoroughly tested before it can be deployed to distributed ledgers, there might be some cases where a deployed smart contrat is found to have some flaws. To fix such flaws, redaction operations can be used to change the content of the deployed smart contract.
* **Smart Contract Upgrading:** After a smart contact is deployed to distributed ledgers, there might be some needs to introduce new functions to the smart contract. Instead of creating a new smart contract, redaction operations can be used to update the existing smart contact with the needed new functions.

## 5.4 Data Sharing

A data provider may publish its data to distributed ledgers so that the data can be used by data consumers. For example, once a piece of data has been published and stored on distributed ledgers, any authorized data consumer can access and retrieve the data directly from distributed ledgers.

Redactable distributed ledgers can be used to :1) remove a piece of on-chain data; and 2) modify a piece of on-chain data

* **Data Removal:** When the data is published for the first time, there might not be any privacy concern. However, any published data could cause privacy concern at a later time. Furthermore, the data provider should have “the right to be forgotten” according to GDPR. Thus, redaction operations can be used to remove any published data from distributed ledgers. For example, if the published data is included in a transaction, it is recommended that this transaction is deleted from distributd ledgers. If all data included in a block needs to be removed, it is recommended that the whole block is deleted.
* **Data Modification:** The published data could have some errors or the data publisher needs to publish a new version of the data. For such scenarios, the data publisher can use redaction opeations to directly modify previously pubished data, instead of publishing the data in new transaction(s) or block(s) and leaving previous data still stored in distributed ledgers.

# 6 Examples of Redactable Distributed Ledgers

## 6.1 Introduction

This clause describes some existing examples for redactable distributed ledgers, especially redactable blockchain. Those existing solutions include TCH-based redaction, policy-based redaction, and redaction based on new block structures.

## 6.2 TCH-based Redactable Blockchain

### 6.2.1 Trapdoor-Controlled Hash

Trapdoor-Controlled Hash (TCH) uses a pair of a Trapdoor Key (TK) and a Public Key (PK) to cause a hash collision between two different messages (e.g., an old message m1 and a new message m2) as follows,

* Hash(m1, PK, OldPublicParameters) = Hash(m2, PK, NewPublicParameters) (1)
* NewPublicParameters = Function(TK, m2, PK, OldPublicParameters) (2)

where Hash() is the function to calculate the hash value of a given message and Function() is another function to calculate NewPublicParamters using the TK, the PK, and OldPublicParameters. It should be computationally challenging for any party to derive NewPublicParameters by brute-force approaches based on Function(), without knowing the TK; otherwise, PDL loses its immutability property.

Essentially, there are two steps in TCH to introduce a hash collision:

* Step 1: find NewPublicParameters for the new message m2 according to equation 2 using the TK; and
* Step 2: calcaute the hash value of the message m2 using Hash(m2, PK, NewPublicParameters), which will be equal to the hash value of the old message m1 (i.e., Hash(m1, PK, OldPublicParameters), according to equation 1.

### 6.2.2 Blockchain Redaction Process

Assume m1 be the content of an old block and its content needs to be modified to m2. Use equations 1 and 2 guarantee that the hash value of this block’s new content (i.e., m2) remains the same; in other words, the hash-based chain structure of blocks is still maintained.

**Figure 6.2.2-1** illustrates two blocks i and i+1 before performing any redaction operation. As it shows, the PrevBlockHash in Block i+1 is the hash value of the content of Block i (i.e., m1), which is equal to Hash (m1, PKi, OldPublicParameters). Note that Block i now contains at least two extra fields (i.e., PKi and OldPublicParameters), compared to traditional blockchain without supporting redaction operation.



Figure 6.2.2-1: Blockchain before a Redaction Operation

**Figure 6.2.2-2** illustrates an example of redaction operation, where the content of Block i is modified from m1 to m2, while maintaining the same hash (i.e., PrevBlockHash) in Block i+1.



Figure 6.2.2-2: Blockchain after a Redaction Operation (i.e., change the content of Block i from m1 to m2)

### 6.2.3 Blockchain Redaction Management

According to **Figure 6.2.2-2**, any party who knows the TK can change the content of Block i to any arbitrary value m2 and in turn use m2 and the TK to find NewPublicParameters. Then, this party simply replaces “OldPublicParameters” with “NewPublicParameters” as contained in Block i. As a result, the change of the content of Block i from m1 to m2 does not introduce any change to next Block i+1, since PrevBlockHash in Block i+1 does not change. In other words, any party can easily modify the content of any block if this party knows the corresponding TK.

Blockchain redaction management refers to the management of the TK, which is extremely critical to prevent unauthorized parties from modifying block content and impairing PDL immutability. There are two approaches to manage the TK: Centralized Trapdoor Key Management and Decentralized Trapdoor Key Management.

* Centralized Trapdoor Key Management: In this setting, only an authorized party knows and maintains the TK. PDL governance appoints this authorized party. As a result, no other party will be able to modify block content (without reverting to bruce-force approaches) and PDL maintains its immutability property.
* Decentralized Trapdoor Key Management: In this setting, the original TK is divided into n TK shares (e.g., TK-1, TK-2, …, TK-n) so that the original TK can be derived from any k(<n) out of n TK shares, where n and k are intergers larger than 1. Each of those n TK shares will be assigned to and held by a different authorized party. PDL governance appoints those n authorized parties. In order to perform a redaction operation, at least k authorized parties need to collaboratively exchange their TK shares to recover the original TK; then any of those k parties can use equation 2 (i.e., Function()) to modify block content.

## 6.3 Policy-based Redactable Blockchain

Mutable Blockhain has been proposed in [i.7], which basically stores multiple different versions of a transaction in the blockchain. Each version represents a transaction containing different content. A user can designate which version should be used by the blockchain system to derive the global state of the whole ledger system. It works as follows:

* A sender first sends multiple pre-defined versions of a transaction (i.e., a transaction set) to validators/minors and indicates one version as the active transaction, which will be used by the validators to derive the global state. The sender also sends a mutability policy to the validators, which describes the conditions for replacing the active transaction with another version of the transaction; for instance, a mutability policy could indicate which party (i.e., a mutator) can replace the active transaction and during which time window.
* Validators validate the active transaction. They also generate a merkle tree based transaction root for all different versions of transactions contained in the transaction set. Then, the active transaction becomes avaiable to the recipient and other regualr users; in other words, the validators use the active transaction to derive/update the global state.
* A mutator sends another transaction (referred to as mutant transaction in [i.7]) to the validators to request replacing the active transaction Ta with another existing transaction Tx (i.e., a different version) contained in the transaction set Tset. The mutant transaction only needs to indicate a reference to the transaction set Tset and a reference to the exising transaction Tx.
* After receiving the request from the mutator, the validators identify Tset and validate Tx. Then, the validators verify if the mutator is allowed to replace Ta with Tx, according to the mutability policy associated with Tset. If the mutator is allowed, the validators will record Tx as the currently active tranaciton for Tset and use Tx to derive the new global state. Dependent on the content of Tx, the validators may need to transmit Tx to the original recipient of Ta.

In fact, this solution does not physically “modify” blockchain itself, but just dynamically changes the active transaction of a transaction set which has been stored in the blockchain. Validators always use the active transaction as the current view about the transaction set and use it to update the global state. Since a transaction has multiple versions being stored in the blockhain, this approach requires more storage. It is noted that [i.7] has not discussed whether the mutability policy can be updated or not.

Another policy-based block redaction for permissionless blockchain systems was proposed in [i.8]. A redaction policy specifies requirements and conditions for approving a redaction operation (e.g., to edit/update an old transaction). As an example, a redaction policy could indicate: 1) only unspendable data of an old transaction can be edited (e.g., removed); and 2) the redaction needs to receive more than 50% votes from all miners. When a user needs to redact an old transaction using a new candidate transaction, it first creates a special transaction containing the identifier of the old transaction and the identifier of the candidate transaction; then the user broadcasts both the special transaction and the candidate transaction to the system; miners will receive and validate the candidate transaction by comparing it to the old transaction and checking the redaction policy; a miner can vote for the redaction request by simply including the hash of the redaction request to the header of next block to be created; after the voting period, everyone can verify if sufficient votes according to the redaction policy have been received; then, a miner can start to replace the old transaction with the candidat transaction (i.e., to replace the old block with a new block). In order for other miners to validate the new block, the approach proposed in [i.8] introduced that each block header keeps two merkle roots: 1) “old\_merkle\_root” is calculated based on all old transactions; and 2) “merkle\_root” is calculated based on old and new transactions.

## 6.4 Redaction Using New Block Structures

BlockMatrix [i.9] is a new data structure that mimics the concept of the blockchain but in a different structure. By definition, BlockMatrix is a type of data structure, which organizes a set of blocks in a matrix and enables both data integrity and data editing. Each block in BlockMatrix sits in a cell with a column index and a row index, which means each block within the matrix is protected by two hashes (i..e, a column hash and a raw hash). When a block is generated and needs to be inserted into the matrix, BlockMatrix has an algorithm to determine the column index and the row index for the new block, so that any two consecutive blocks will not be placed in the same column or in the same row. To protect block integrity, BlockMatrix does not need to calculate the hash of every single block but calculates a hash value for all blocks in the same column (i.e., a column hash) and a hash value for all blocks in the same row (i.e., a row hash). As a result, when a new block is added to the matrix, a new column hash and a new row hash need to be re-calculated. Similarly, when an existing block needs to be edited or deleted, the corresponding column hash and the row hash will be re-calculated. BlockMatrix is more applicable for permissioned distributed ledgers instead of permissionless ledgers.

## 6.5 Discussions

Existing redactable distributed ledgers introduce different complexities in terms of communication, computation, and storage. In addition, they have different security levels and could be applied to permissioned ledgers, permissionless ledgers, and/or both. The pros and cons of existing redactable distributed ledger examples are briefly summarized below.

TCH-based redactable blockchain generally has very low storage overhead. It also does not introduce high communication or computation overhead. However, it needs to carefully manage trapdoor keys in order to control which parties can redact blocks or transactions. Centralized trapdoor key management could be useful for permissioned ledger systems, while decentralized trapdoor key management is more applicable for permissionless ledger systems. Also, decentralized trapdoor key management introduces extra communication/computation overhead for managing trapdoor keys across multiple distributed parties.

Policy-based redactable blockchain needs to maintain multiple versions of transacitons, which leads to high storage overhead. In addition, extra communication and computation overhead are introduced for exchanging and managing redaction policies. Those policy-based approaches could be more applicable to permissioned ledger systems, but they also can be implemented for permisisoneless ledger systems.

BlockMatrix in [i.9] is just a different ledger structure that can better facilitate redaction operations. Compared to TCH-based or Policy-based redactable blockchain, BlockMatrix does not have stringent control on which parties can redact blocks. As such, BlockMatrix is better to be used together with certain redaction policies for permissioned ledger systems.

# 7 Conclusions and Next Steps

## 7.1 Introduction

The present document discussed redaction distributed ledgers. It first described the concepts including redaction operations. Three use cases that can be benefited from redactable distributed ledgers were described. Then some existing examples of redactable distributed ledgers were presented. Finally, recommendations for next steps are included.

## 7.2 Recommendations for Next Steps

As described in clause 6, there are various solutions of redactable distributed ledgers, which have different characteristics such as communication overhead, computation overhead, storage overhead, etc. Different redactable distributed ledger technologies are still evolving and they may be applicable for different PDL-based applications and systems. As such, it is out of the scope of ISG PDL to standardize a particular redactable distributed ledger technology. However, the following aspects could be considered for standardization by ISG PDL.

* Specifications on the type of redaction operations could be developed.
* Specifications on redaction policies could be developed.
* Specifications on managing redaction operations could be developed.
* Specifications on the functionalities of ETSI-ISG-PDL reference architecture to support redaction operations could be developed.

# History

|  |  |  |
| --- | --- | --- |
| **Document history** | | |
| V0.0.1 | 18 Jul 2022 | Creation of document, TOC and high-level content |
| V0.0.2 | 27 Sept 2022 | PDL(22)000\_125r2 |
| V0.0.3 | 26 Jan 2023 | PDL(22)000\_148r1, PDL(23)000\_011r1 |
|  |  |  |
|  |  |  |