ETSI GR PDL 018 V1.2.1 (2023-10)

Permissioned Distributed Ledger (PDL); Redactable Distributed Ledgers

Disclaimer

The present document has been produced and approved by the Permissioned Distributed Ledger (PDL) 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.

Reference
RGR/PDL-0018v121_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

Contents

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 (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

The present document describes the definition of redactable distributed ledgers, presents use cases where redactable distributed ledgers are useful, and assesses existing solutions of redactable distributed ledgers. The present document also discusses potential standardization areas for enabling, managing, and using Redactable Distributed Ledgers (RDLs).

2 References

2.1 Normative references

Normative references are not applicable in the present document.

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.

[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.

[i.4] IoTA 2.0 Research Specifications, June 4, 2021.

[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.

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

[i.7] ETSI GS PDL 011: "Permissioned Distributed Ledger (PDL); Specification of Requirements for Smart Contracts' architecture and security".

[i.8] I. Puddu, A. Dmitrienko and S. Capkun (2017): "μChain: How to Forget without Hard Forks".

[i.9] 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.

[i.10] K. D. Richard: "A Data Structure for Integrity Protection with Erasure Capability", NIST Cybersecurity Whitepaper, May 20, 2022.

3 Definition of terms, symbols and abbreviations

3.1 Terms

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

hashing collision: scenario where two different input messages get the same hashing value using the same hashing function

Redactable Distributed Ledger (RDL): distributed ledgers where the stored content or objects can be modified with consensus through certain redaction operations

NOTE: A survey on mechanisms for mutable blockchains is presented in [i.1].

redactable objects: objects on distributed ledgers with the redaction property

redaction: property for supporting changes to one or multiple objects on distributed ledgers

redaction operations: actions or operations to change redactable objects on distributed ledgers

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

Trapdoor-Controlled Hash (TCH): 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, as described in [i.2], is an example of trapdoor-controlled hash schemes.

Trapdoor Key (TK): 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:

BEL	Blockchain Edit List
CH	Chameleon Hash
DAG	Directed Acyclic Graph
DAO	Decentralized Autonomous Organization
DLT	Distributed Ledger Technology
ETSI	European Telecommunications Standards Institute
GDPR	General Data Protection Regulation
ID	IDentifier
ISG	Industry Specification Group
PDL	Permissioned Distributed Ledger
PK	Public Key
PKi	Public Key i
RDL	Redactable Distributed Ledger
RSA	Rivest–Shamir–Adleman
TCH	Trapdoor-Controlled Hash
TS	TimeStamp
TK	Trapdoor Key

TXN Transaction

4 Introduction to Redactable Distributed Ledger

4.1 Introduction

Distributed Ledger Technology (DLT) systems have been evolving.

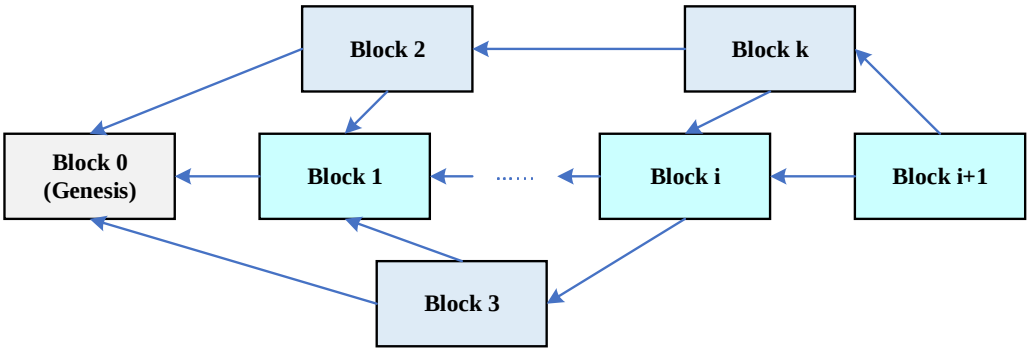
EXAMPLE 1: Distributed ledgers can be formed in different structures as illustrated in Figure 4.1-1, such as blockchain (e.g. Bitcoin™, Ethereum™, Hyperledger Fabric™), block Directed Acyclic Graph (DAG) (e.g. PPHANTOM as described in [i.3]), and blockless DAG (e.g. IoTA as described in [i.4]). In general, DLT brings unique characteristics and advantages such as immutability, transparency and decentralization.

1) **Blockchain:** A linear topology with a set of chained blocks starting from the first genesis block. Each block (except the genesis block) has one and only one parent block.

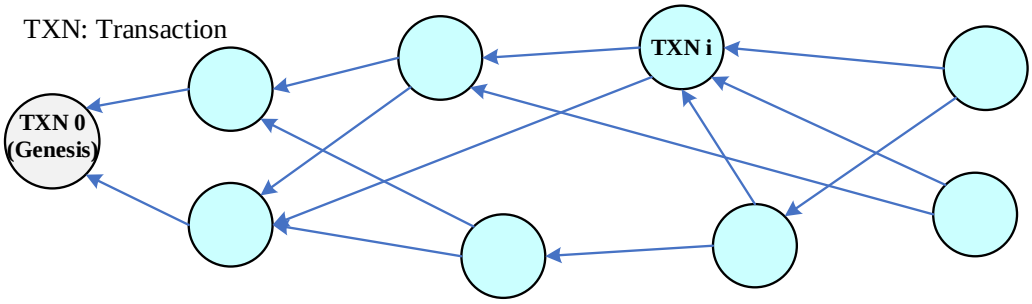
2) **Block DAG:** Blocks are organized in a DAG, where each node represents a block. Usually, each block (except the genesis block) has more than one parent blocks. Two or more blocks are connected if the corresponding two nodes are connected in the DAG.

3) **Blockless DAG:** Transactions are directly organized in a DAG, where each node represents a transaction. Usually, each transaction (except the genesis transaction) has more than one parent transactions. Two or more transactions are connected if the corresponding two nodes are connected in the DAG.

(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.

EXAMPLE 2: 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 Limitations of Immutable Ledgers

The unique characteristics of DLT (especially immutability) could be misused and lead to potential issues. Figure 4.2-1 illustrates some potential limitations of immutable ledger structure:

1) Information published by users on distributed ledgers may become sensitive and create privacy concerns in the future, especially in public DLT systems. Such privacy-concerned information cannot be removed from distributed ledgers due to their immutability, therefore contradicting "the right to be forgotten" associated with General Data Protection Regulation (GDPR).

2) Misinformation could be added to the distributed ledgers by attackers and stay there forever.

3) Crypto criminals and hackers can inject illegal contents forbidden by national or international laws into distribute ledgers, which cannot be removed.

4) Bogus smart-contracts, and more specifically bogus Decentralized Autonomous Organization (DAO) smart-contracts, can be exploited and immutability limits the ability to rectify such problems. E.g. DAO applications, the most significant smart contract applications in the Ethereum™ platform, are another example of immutability misuse. Hackers and crypto criminals discovered logical flaws and vulnerabilities in DAO smart contracts that led to transferring over US$ 120,3 million worth of Ethereum™ coins to their accounts as reported in [i.5], which could have been avoided if such flawed smart contracts had been modified. This problem was semi-cured by hard forking Ethereum™ blockchain back in 2016 to delete the attackers' transfer transactions.

5) Finally, immutability unavoidably causes a scalability issue in maintaining the append-only and ever-growing chain-length of distributed ledgers.

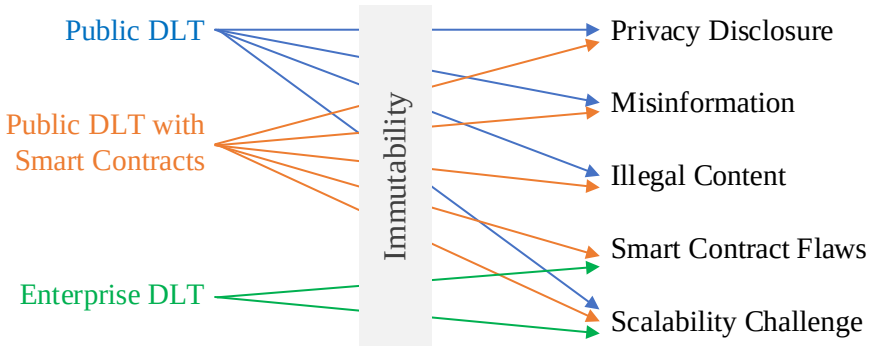


Figure 4.2-1: Potential Limitations with Immutable Distributed Ledgers

4.3 Redaction Operations

4.3.1 Introduction

Redaction operations could be different for different structures of distributed ledgers,. Even if it is possible to apply the same redaction operation on different distributed ledger structures, the complexity and implications of the redaction operation could still be different.

4.3.2 Redaction Operations for Blockchain

4.3.2.1 Types of Blockchain Redaction Operations

Redaction operations for blockchain-based distributed ledgers are: transaction-level redaction operations, single-block-level redaction operations, and multiple-block-level redaction operations.

4.3.2.2 Transaction-Level Redaction Operations

4.3.2.2.1 Changes Caused by Transaction-Level Redaction Operations

These redaction operations aim to impose changes on one or multiple existing transactions being included in an existing block and/or 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
3) the hash value of this block.

4.3.2.2.2 Scope of Transaction-Level Redaction Operations

Transaction-level redaction operations include:

1) modification of an existing transaction, which could be any one or multiple fields of the existing transaction;
2) removal of an existing transaction from the corresponding block;
3) insertion of a new transaction to an existing block;
4) modification of more than one existing transaction from the corresponding block;
5) removal of more than one existing transactions from the corresponding block; and
6) insertion of multiple transactions to an existing block.

4.3.2.3 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. Single-blockchain-level redaction operations include:

1) modification of non-transaction-related fields of an existing block;
2) removal of an existing block; and
3) insertion of a new block to an existing blockchain (not to append the new block to the existing blockchain).

4.3.2.4 Multiple-Blocks-Level Redaction Operations

These redaction operations are used to introduce changes related to multiple blocks, such as:

1) removal of multiple existing consecutive blocks;
2) removal of multiple existing non-consecutive blocks, which will essentially be using "Single-Block-Level Redaction Operations";
3) insertion of multiple new blocks in a consecutive order to an existing blockchain (not to append blocks to existing blockchain); and
4) insertion of multiple new blocks in a non-consecutive order to an existing blockchain (not to append blocks to existing blockchain).

4.3.3 Redaction Operations for Block DAGs

4.3.3.1 Types of Block DAG Redaction Operations

Redaction operations for a block DAG are similar to redaction operation to a blockchain and include the following: transaction-level redaction operations, single-block-level redaction operations, and multiple-block-level redaction operations.

4.3.3.2 Transaction-Level Redaction Operations

They are the same as transaction-level redaction operations for a blockchain as descried in clause 4.3.2.2.

4.3.3.3 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:

1) modification of non-transaction-related fields of an existing block;
2) removal of an existing block; and
3) insertion of a new block to the existing block DAG (not to append the new block to the existing block DAG).

4.3.3.4 Multiple-Blocks-Level Redaction Operations

These redaction operations are used to introduce changes related to multiple blocks, such as:

1) removal of multiple existing connected blocks;
2) removal of multiple existing non-connected blocks;
3) insertion of multiple new blocks in a connected sub-graph to an existing block DAG (not to append blocks to existing block DAG); and
4) insertion of multiple new blocks in a non-connected way to an 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, when changing an existing transaction or inserting a new transaction any impact and implications to its child nodes and/or parent nodes needs to be considered and addressed.

The following redaction operations are possible for a blockless DAG:

1) modification of an existing transaction, which could be any one or multiple fields of the existing transaction;
2) removal of an existing transaction from the current blockless DAG;
3) insertion of a new transaction to the current blockless DAG;
4) modification of multiple connected transactions from the current blockless DAG;
5) removal of multiple connected transactions from the corresponding block; and
6) insertion of multiple connected transactions to the current blockless DAG (not to append them as leaf nodes).

5 Use Cases for Redactable Distributed Ledgers

5.1 Introduction

Redactable distributed ledgers are useful for some PDL-based applications such 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 identifiers on distributed ledgers. The user requests and obtains its credentials from a third-party issuer. 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 additional information from the issuer. The verifier may obtain public information about the user (e.g. the public key of the issuer), and can verify and authenticate the credentials directly from distributed ledgers.

NOTE: Such credential public information has also been stored on distributed ledgers.

Redactable distributed ledgers can be used to:

1) **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.

2) **Modify Credential Public Information:** Credential public information describes how a corresponding credential should be verified. Since credential public information is stored on distributed ledgers, it can be accessed by any authorized entities (including verifiers). The way to verify and authenticate a credential could be changed; as a result, the credential public information stored on distributed ledgers needs to be modified accordingly. Redaction operations can be used for this purpose.

3) **Remove an Identifier:** User identifiers are public information 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 [i.7] are auto-executable code or program 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 cannot 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:

1) **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 may be cases where a deployed smart contract is found to have flaws. To fix such flaws, redaction operations can be used to change the content of the deployed smart contract.

2) **Smart Contract Upgrading:** After a smart contact is deployed to distributed ledgers, there may be a need to introduce new functions to the smart contract. Instead of creating a new smart contract, redaction operations can be used to upgrade the existing smart contact to include the new functionalities.

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 the distributed ledgers.

Redactable distributed ledgers can be used to remove a piece of on-chain data; and modify a piece of on-chain data as described below:

1) **Data Removal:** When the data is published for the first time, there may not be any privacy concern. However, any published data could cause privacy concern at a later time. Furthermore, in scenarios where GDPR needs to be observed the data provider should offer "the right to be forgotten". Thus, redaction operations can be used to remove any published data from distributed ledgers. E.g. if the published data is included in a transaction, it is recommended that this transaction is deleted from distributed ledgers. If all data included in a block needs to be removed, it is recommended that the whole block is deleted.

2) **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 operations to directly modify previously published data, instead of publishing the data in new transaction(s) or block(s) and leaving previous data still stored and accessible in distributed ledgers.

5.5 Ledger Size Reduction

A key feature of traditional blockchain system is that a full history is stored in the ledger on every participating node of the system. Though this is helpful to accountability and traceability for the ledger data, it significantly consumes the local storage capacity of a blockchain node. Particularly, due to the heterogeneity of blockchain nodes, the storage capacities are different. For example, a server machine node certainly has higher storage capacity than an end device. Hence, it is important for a blockchain system to be able to have the capability of ledger size reduction. There are two sub-cases as described below:

1) **Node Ledger Size Reduction:** If only one or multiple nodes have to reduce the size of their local ledger data, some history data is able to cut off from the entire ledger history. Meanwhile, for those nodes with sufficient storage capability or responsible for maintaining the full data history, they can still restore the entire ledger data. In this case, the traceability and verifiability should be remained in the blockchain system.

2) **System-wide Ledger Size Reduction:** If the whole blockchain system agrees that some part of the ledger data should be removed from the history, then every node is able to make such a change with a specified blockchain redaction method to reduce the ledger size.

6 Examples of Redactable Distributed Ledgers

6.1 Introduction

This clause offers examples of redactable distributed ledgers, including redactable blockchains. Those existing solutions include TCH-based redaction, policy-based redaction, and redaction based on new ledger structures. More examples of redactable distributed ledgers can be found in [i.2].

6.2 TCH-based Redactable Blockchains

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:

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

Where Hash() is the function that calculates the hash value of a given message and Function() is another function that calculates NewPublicParameters 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. calculate 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 (Hash(m1, PK, OldPublicParameters)), according to equation 1.

6.2.2 Blockchain Redaction Process

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

Figure 6.2.2-1 illustrates two blocks i and i+1 before performing any redaction operation. The PrevBlockHash in Block i+1 is the hash value of the content of Block i (m1), which is equal to Hash (m1, PKi, OldPublicParameters).

NOTE 1: Block i now contains at least two extra fields (PKi and OldPublicParameters), compared to traditional blockchain without supporting redaction operation.

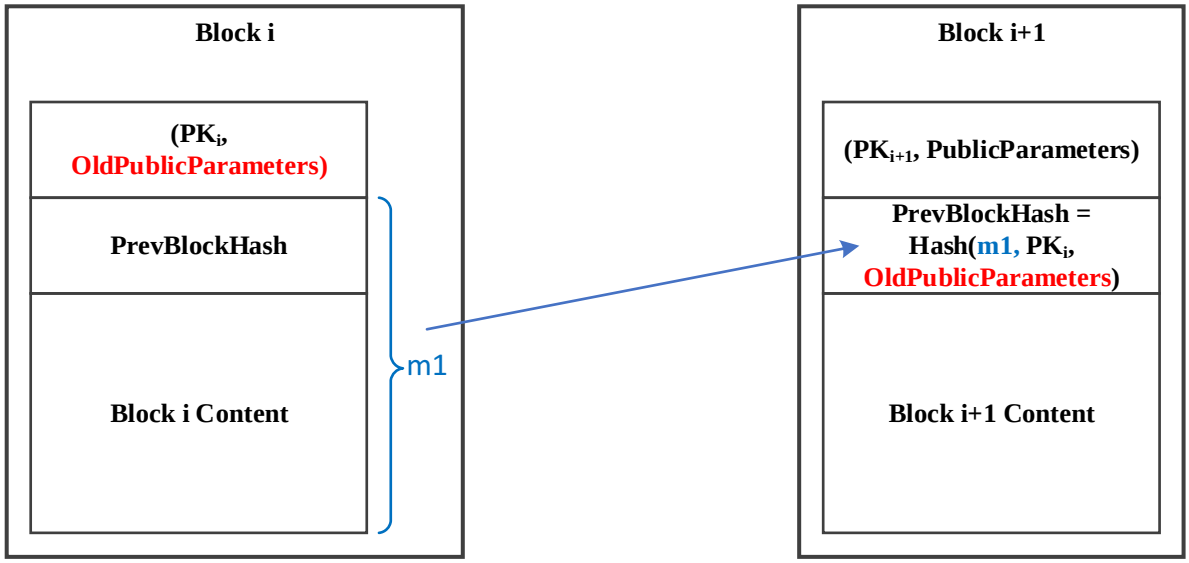


Figure 6.2.2-1: Blockchain before a Redaction Operation

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 (PrevBlockHash) in Block i+1.

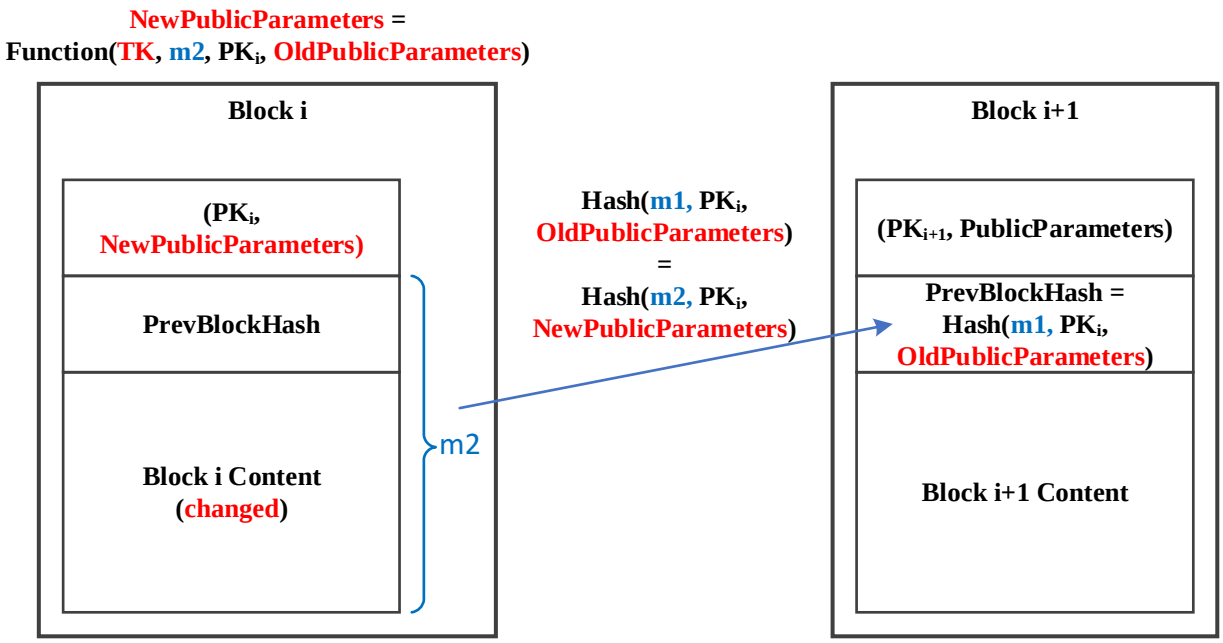


Figure 6.2.2-2: Blockchain after a Redaction Operation (i.e. change the content of Block i from m1 to m2)

The blockchain system can select a particular hash algorithm if it identifies that a transaction would be modified later; in addition, a redactable blockchain system can also select a set of blockchain nodes that own the public key for the chameleon hash function and are responsible for transaction redaction. Specifically, when a transaction is composed and mined to be published on the blockchain, specified blockchain nodes that are responsible for transaction redaction can calculate the hash value for the transaction header with the corresponding public key of the chameleon hash function. Later on, when the transaction has to be modified, those blockchain redaction nodes will calculate a new hash collision with the private key of the chameleon hash function owned by those nodes.

NOTE 2: During and after the calculation, the private key parts will not be exposed.

6.2.3 Blockchain Redaction Management

According to Figure 6.2.2-2, any party that 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. This party can then replace "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 crucial 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:

1) **Centralized Trapdoor Key Management:** When implementing this approach, only an authorized party knows and maintains the TK. PDL governance can appoint 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.

2) **Decentralized Trapdoor Key Management:** When implementing this approach, 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 integers larger than 1. Each of those n TK shares will be assigned to and held by a different authorized party. PDL governance can appoint 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 (Function()) to derive NewPublicParameters and be able to modify the block content.

6.2.4 Blockchain Redaction Verification

Because the CH value of the edited block remains unchanged, the client cannot distinguish the latest state of the block. Therefore, the CH-based redactable blockchain needs to have the verifiability of editing history. The blockchain can provide clients with the function of checking editing history, verify the validity of blocks through consistency check, and whether blocks have been modified.

The editable history of blockchain can use RSA accumulator, which is a one-way member hash function. It allows users to prove that potential elements are members of a set without revealing individual members in the set. In a CH-based redactable blockchain, the value of RSA accumulator is included in the block header. When a new block is modified, the hash of the modified block and the value of the accumulator are used to calculate the latest value of the accumulator, which is carried in the latest block header to update the value of RSA accumulator. When a client sends a query message about the height value of a block to blockchain node, blockchain node returns the client the queried block and a proof of membership or non-membership of the block. The two cases of its membership as follows:

1) Membership Witness: To prove that an element is in the set, that is, the block has been edited.

2) Non-membership Witness: To prove that an element is not in the set, that is, the block has not been edited.

After the client receives the membership or non-membership witness of the block, it verifies the transactions in the block and the validity of the block. If it is valid, the client can complete the consistency check of the block by verifying the membership/non-membership witness

6.3 Policy-based Redactable Blockchains

6.3.1 Pre-defined Mutability

Mutable Blockchain has been proposed in [i.8], 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:

1) A sender first sends multiple pre-defined versions of a transaction (e.g. 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 (e.g. a mutator) can replace the active transaction and during which time window.

2) 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 available to the recipient and other regular users; in other words, the validators use the active transaction to derive/update the global state.

3) A mutator sends another transaction (referred to as mutant transaction in [i.8]) to the validators to request replacing the active transaction Ta with another (existing, pre-defined) transaction Tx that is already 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 existing transaction Tx.

4) 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 transaction 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.

This method does not modify the blockchain itself. It dynamically changes the active transaction to another transaction that has already been stored in the blockchain. Validators always use the currently active transaction to update the global state. Since multiple versions of transactions are stored in the blockchain, this approach increases the size of the chain thus requiring more storage and more computational resources. It is noted that the authors of [i.8] have not discussed whether the mutability policy can be updated or not.

This method only allows selection between pre-defined options. It does not allow adding new transaction options after the initial transaction had taken place.

6.3.2 Voting-based Mutability

6.3.2.1 Definition

Another policy-based block redaction for permissionless blockchain systems was proposed in [i.9]. 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.

NOTE: Dependent on the governance, a redaction policy can designate other types of consensus methods (e.g. less than 50 % votes).

6.3.2.2 Method

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 node can vote for the redaction request by including the hash of the redaction request in the header of next block to be created; after the voting period, if sufficient votes according to the redaction policy have been received, nodes replace the old transaction with the candidate transaction (e.g. replace the old block with a new block). In order for nodes to validate the new block, the approach proposed in [i.9] requires 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 Ledger Structures for Redactable Blockchains

6.4.1 BlockMatrix

BlockMatrix, as described in [i.10], is a new data structure that deviates from the concept of blockchain and uses a different structure. BlockMatrix is a data structure that places a set of blocks in a matrix and enables both data integrity and data editing. Each block in a BlockMatrix sits in a cell defined by a column index and a row index, which means each block within the matrix is protected by two hashes (a column hash and a row hash). When a block is generated and inserted into the matrix, a BlockMatrix uses an algorithm to determine the column index and the row index for the new block with a restriction 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 (a column hash) and a hash value for all blocks in the same row (a row hash). As a result, when a new block is added to the matrix, a new column hash and a new row hash are re-calculated. Similarly, when an existing block is edited or deleted, the corresponding column hash and row hash will be re-calculated. BlockMatrix is more applicable for permissioned distributed ledgers rather than permissionless distributed ledgers.

6.4.2 Triple-Hash Block Structure

To ensure the traceability of transactions, old blocks (or old block header information) can be selectively retained in the blockchain. The old and the edited blocks together form an editing chain in the blockchain and are maintained at the same height in the blockchain. Therefore, the blockchain forms two dimensions in the horizontal and vertical directions. The horizontal direction is the growth direction of blocks, and the vertical direction is the editing direction of blocks. Each block in the vertical direction has the same header hash.

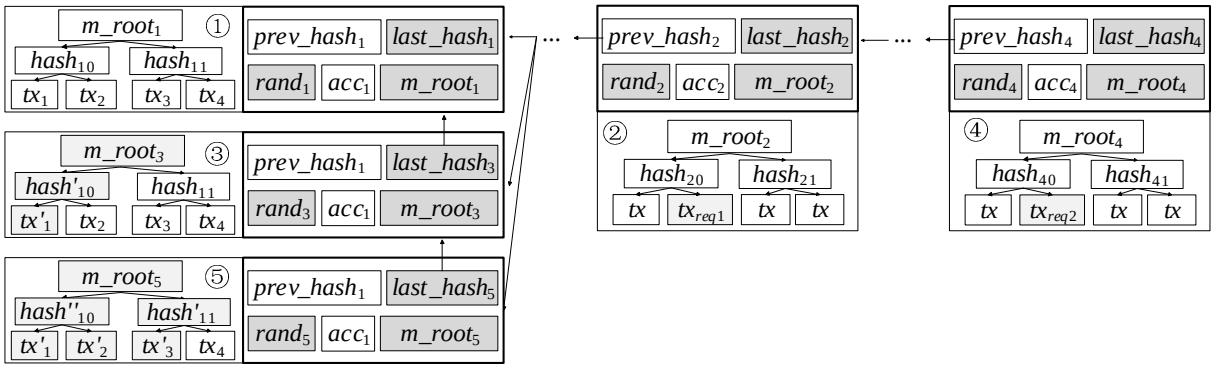


Figure 6.4.2-1: Triple-Hash Blockchain

Different from existing non-redactable blockchain structure, a triple-hash block header is extended with the following fields:

- *prev_hash*: This field stores the hash value of the previous block in the horizontal direction. This hash value is calculated through a (decentralized) chameleon hash function. In traditional blockchain systems, all block headers of different heights form a chain through this field.
- *last_hash*: This field stores the traditional hash value (such as SHA256) of the block that the edited block has been edited from. It records the header hash value of the previous block in the vertical direction. All block headers of the same height form an editing chain through this field.
- *acc*: This field stores the digest of the RSA accumulator. The accumulator encodes the last_hash values in all blocks. *acc* is used to verify whether a block has been edited.
- *Rand:* This field stores the random value used by the chameleon hash function to ensure that the old block and the new edited block can be at the same height, that is, they have the same chameleon hash value.

6.4.3 Two-layer Hash Block Structure

In addition to Triple-hash block structure, another option is a two-layer hash blockchain structure. Different from a triple-hash block structure, a two-layer hash block structure also uses chameleon hash function but directly at the level of Merkel tree, instead of applying at the prev_hash field as in the triple-hash block structure.

Specifically, for the root of the Merkel tree organizing all transactions in the block, when any change happens, e.g. a transaction data is modified, the Merkel tree root, which is also a hash value, will be re-calculated with a selected chameleon hash function. In this way, the header fields will be intact so the whole blockchain structure remains unbroken because all hash values in the header still use traditional hash, which is immutable.

In addition to the traditional fields, as shown in Figure 6.4.3-1, a two-layer block structure adds a new field called Blockchain Edit List hash (BEL_hash). BEL_hash is used to record all relevant information about modifications to the block including the block ID, transaction identifier, timestamp, modification type and so on. For composing a block, a blockchain node will calculate the hash value for BEL and put the BEL_hash into the block header. This guarantees that every modification can be tracked with the BEL and its integrity can be checked with the BEL_hash value. Besides, the modified block structure contains two kinds of timestamps. The first kind is the timestamp of the whole block, which records the time when the block is first created; the second kind is a TimeStamp (TS) associated with the hash value of the Merkel Tree organizing all the transactions in the block. The TS aims to record the time when the transaction information is modified. This is different to the timestamp of the whole transaction block.

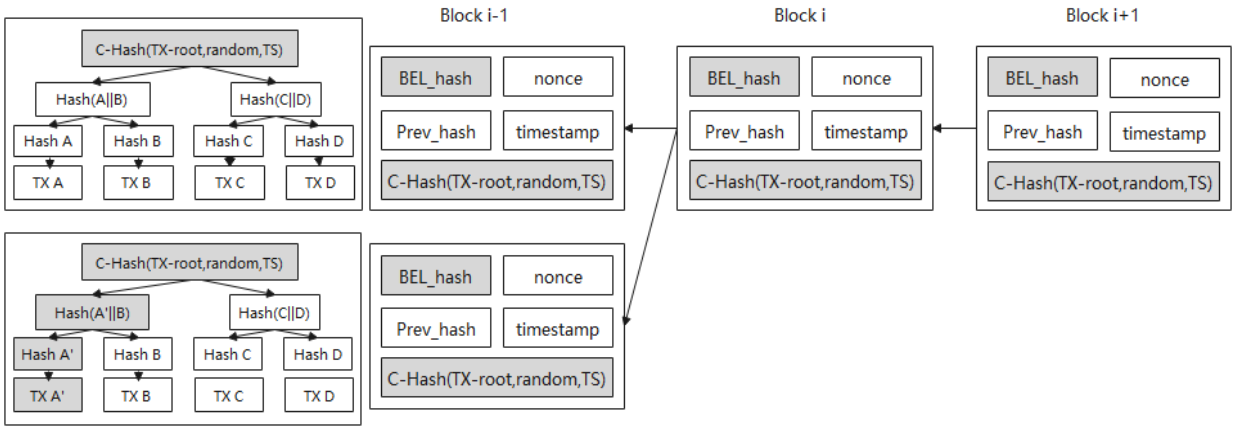


Figure 6.4.3-1: Two-layer Block Structure

6.5 Discussions

Existing redactable distributed ledgers as described in clauses 6.2, 6.3, and 6.4 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 blockchains generally have very low storage overhead. They also do not introduce high communication or computation overheads. However, trapdoor keys have to be carefully managed 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. Note that decentralized trapdoor key management introduces additional communication/computation overheads for managing trapdoor keys across multiple distributed parties.

Policy-based redactable blockchain in [i.8] needs to compute and maintain multiple versions of transactions, which leads to high computation and storage overheads. In addition, extra communication and computation overheads are introduced for exchanging and managing redaction policies. Policy-based approaches such as pre-defined mutability in [i.8] and voting-based mutability in [i.9] would be more applicable to permissioned ledger systems, but they also can be implemented on permissionless ledger systems given sufficient computational and storage resources are available.

BlockMatrix, as described in [i.10], uses a different ledger structure that is more suited to facilitate redaction operations. Compared to TCH-based or Policy-based redactable blockchain, BlockMatrix does not have stringent control (either through consensus or other methods) as to which parties can redact blocks. As such, BlockMatrix would likely be deployed in tandem with redaction policies in permissioned ledger systems rather than in a permissionless environment that lacks change-management and control.

NOTE: Existing redactable distributed ledgers are mainly designed for blockchain-based ledgers; it is unclear at this point if they can be applied to distributed ledgers based on block DAG and blockless DAG.

7 Conclusions and Next Steps

7.1 Introduction

The present document discusses redaction distributed ledgers. It first describes the concepts including redaction operations. Three use cases that can benefit from redactable distributed ledgers are described. Examples of existing redactable distributed ledgers are presented. Recommendations for next steps are listed in clause 7.2.

7.2 Recommendations for Next Steps

As described in clause 6, there are various possible implementations of redactable distributed ledgers, which have different characteristics such as communication overhead, computation overhead, storage overhead, etc. Additional redactable distributed ledger technologies are evolving that may be applicable to different PDL-based applications and systems. It is beyond of the scope of ETSI ISG PDL to standardize particular redactable distributed ledger technologies. However, the following concepts and topics could be considered for standardization by ETSI ISG PDL:

1) Specifications of the type of redaction operations.

2) Specifications of redaction policies.

3) Specifications of redaction operations management.

4) Specifications of the ETSI-ISG-PDL reference architecture functionalities that are required to support redaction operations.

History

Document history		
V1.1.1	April 2023	Publication
V1.2.1	October 2023	Publication