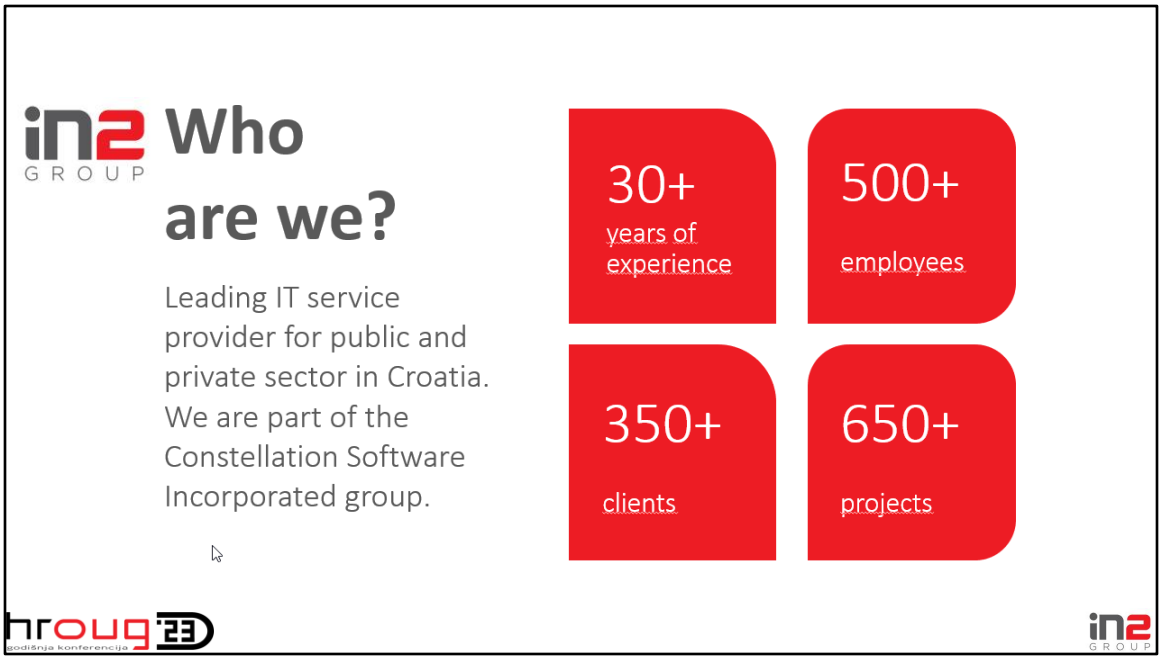
in2 GROUP

Tomislav Kušanić

There is more to blockchain than

hroug '23
godišnja konferencija


in2 GROUP
Who are we?
Leading IT service provider for public and private sector in Croatia. We are part of the Constellation Software Incorporated group.
30+
years of experience
500+
employees
350+
clients
650+
projects
hroug 23
godišnja konferencija
in2 GROUP

About me

- Tomislav Kušanić
  - tomislav.kusanic@in2.hr
- 15 years of experience with Oracle (E-Business Suite, Apex, BI)

Todays topic

- Introduction to Blockchain
- Blockchain tables in Oracle DB

Introduction to Blockchain

- Digital ledger shared accross network of computers
- Anything of value can be tracked and traded on blockchain network
- Each block in chain has unique digital signature
- Blocks are connected in chronological order to form a chain
- Resistant to tampering and hacking

Digital ledger shared accross network of computers
- all participants have access to digital ledger
- immutable record of all transactions
- no need for central authority
- transactions recorded only once

No need for central authority
Anything of value can be tracked and traded on blockchain network
Each block in chain has unique digital signature
Blocks are connected in chronological order to form a chain
Resistant to tampering and hacking

Introduction to Blockchain

- Immutable records
  - no participant can change or tamper with transaction in ledger
  - original transaction cannot be changed or deleted
  - correction is done with correcting transaction

Introduction to Blockchain

- Smart contracts
  - simple programs stored on blockchain
  - run when conditions are met
  - automate execution of an aggreement
  - participants are immediately certain of outcome
  - no intermediary involved or time lost
  - can automate workflow by triggering next action

Use Cases

1. Decentralized record keeping/audit trail
2. Supply chain provenance & authenticity accross trading community
3. Multiparty exchange transactions:
    1. payments
    2. funds-transfer
    3. asset tokenization
4. Digital identity or certifications across multiple issuers
5. Business transactions based on multi party object/document matching & reconciliation
6. Multi-brand loyalty systems

Blockchain in Oracle Database

- Blockchain tables introduced in 21c
- Backported to 19c
- New functionality in 23c

Blockchain tables

- Insert only tables
- Organize rows into chains
- Each row chained to previous row in the chain
- Row tampering changes its hash value indicating data manipulation
- Optional user signatures

Insert only tables that organize rows into chains
Each row is chained to previous row in the chain using cryptographic hash
Row cryptographic hash is based on row data and hash of previous row
Tampering with a row changes its hash value, affecting the subsequent rows in the chain indicating data manipulation
Optional user signatures can be added for enhanced fraud protection, requiring digital certificates.

Blockchain tables

- Indexed and partitioned
- Controlled for dropping
- Rows can be selectively deleted or retained
- Used in transactions and queries alongside regular tables.

Blockchain tables

- Prevent unauthorized modification of data
- Insert-only
- Row retention period

Blockchain tables prevent unauthorized modification of data by insiders or hackers with stolen insider credentials
Table is insert-only
Users cannot delete rows within the defined retention period

Blockchain tables

- Blockchain table definition cannot be changed
- Blockchain table conversion not permitted
- Table data in the database dictionary cannot be modified

Database does not allow users to change the blockchain table definition
Conversion between blockchain table and normal table is not permitted
Table data in the database dictionary cannot be modified

Blockchain tables

- BC table digest generated on request and signed
- Digest based on last row metadata columns
- Data modification results in a change of the digest value

Cryptographic digest of the blockchain table is generated on request and signed with the database schema owner's private key
Digest is based on metadata columns for the last row of every chain in the table
Any data modification results in a change of the digest value

Blockchain tables

- Digest periodically calculated and stored
- Verifying digest for the range of rows between two timestamps

Cryptographic digest can be periodically calculated and stored in distributed safe repositories
Verifying digest for the range of rows between two timestamps detects cover-ups of unauthorized changes

Blockchain tables

- Prevents undetected, unauthorized modifications of data
- End users can sign new row
- Prevent impersonation
- Enables data integrity verification

Prevents undetected, unauthorized modifications of data using stolen end-user credentials
End users can cryptographically sign new row and this confirms end user's role in inserting the record
Digital certificate and private key prevent impersonation and verify data integrity

Blockchain tables

- Blockchain technology is directly integrated into Oracle database
- Utilizes advanced Oracle database functionality
- Minimal changes to existing applications
- No new infrastructure requirements
- Mix blockchain and regular tables in queries and transactions

Blockchain technology is directly integrated into Oracle database for enhanced data protection
Utilizes advanced Oracle database functionality, including analytics on cryptographically secured data
Minimal changes to existing applications and no new infrastructure requirements
Enables users to mix blockchain and regular tables in queries and transactions

Chaining rows

- Row chained to previous row in the chain
- Chain verifiable by all participants
- 32 chains
- Chain unique identifier - instance ID + chain ID

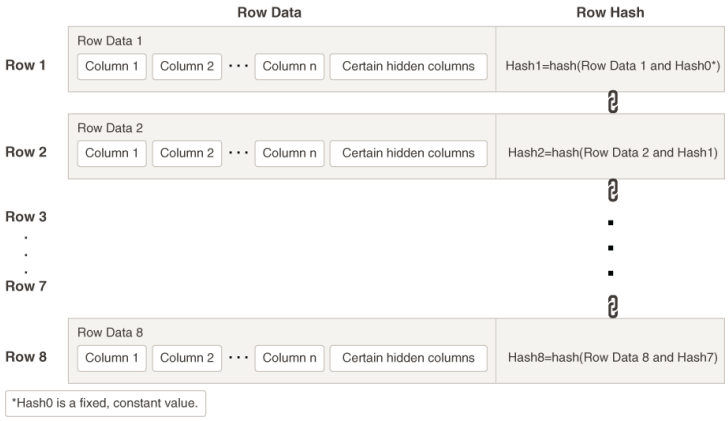
A row in a blockchain table is chained to the previous row in the chain
The chain of rows is verifiable by all participants
Each blockchain table contains 32 chains from 0 to 31
Chains are identified by a unique combination of instance ID and chain ID

Chaining rows

- User columns and hidden columns
- Unique sequence number within the chain
- Each row linked to the previous row

Rows consist of user columns and hidden columns
Upon insertion, a row is assigned a unique sequence number within the chain 1 higher than the previous row's sequence number
Each row is linked to the previous row

Chaining rows



Chaining rows

- Row unique identifier - instance ID + chain ID + sequence number
- SHA-512 hash value computed
- Hash0

Rows can be uniquely identified using instance ID, chain ID, and sequence number
When a row is inserted, a SHA-512 hash value is computed based on row data and the previous row's hash value
The first row uses a fixed, constant value (Hash0) as the previous row's hash

Chaining rows

- Single transaction insert into multiple blockchain tables
- Single transaction insert to the same chain
- Chain row position determined by order of insertion
- Automatic chain selection upon commit

A single transaction can insert rows into multiple blockchain tables
Rows inserted by a single transaction are added to the same chain
The position of rows within the chain corresponds to the order of their insertion into the blockchain table
The database automatically selects the chain for the rows when the transaction commits

Chaining rows

- Parallel transaction row chain position determined by transaction commit order
- Rows linked to the blockchain upon transaction commit
- Higher commit latency
- Avoid inserting very large number of rows in a single transaction

When multiple users insert rows simultaneously into the same chain, the order of adding the rows depends on the transaction commit order
Rows are linked to the blockchain upon transaction commit
Inserting a large number of rows in a single transaction results in higher commit latency
It is recommended to avoid inserting a very large number of rows in a single transaction

Restrictions on blockchain tables

- Following data types not supported:
  - ROWID
  - LONG
  - Object type
  - TIMESTAMP WITH TIME ZONE
  - TIMESTAMP WITH LOCAL TIME ZONE
  - BFILE
  - XMLType

Restrictions on blockchain tables

- Following operations not supported:
  - creating blockchain tables in CDB root
  - updating and merging rows
  - adding, dropping and renaming columns
  - truncating the table
  - dropping partitions
  - inserting data using parallel DML
  - before row triggers for update operations
  - creating ADO, VPD and OLS policies
  - online redefinition
  - converting a regular table to blockchain table and viceversa

Creating blockchain tables

```
CREATE BLOCKCHAIN TABLE transaction_ledger
(transaction_id NUMBER
,transaction_date   DATE
,transaction_user   VARCHAR2(50))
NO DROP UNTIL 8 DAYS IDLE
NO DELETE UNTIL 356 DAYS AFTER INSERT
HASHING USING "SHA2_512" VERSION "V1";
```

Creating blockchain tables

```
CREATE BLOCKCHAIN TABLE transaction_ledger_partitioned
(transaction_id NUMBER
,transaction_date    DATE
,transaction_user    VARCHAR2(50))
NO DROP UNTIL 16 DAYS IDLE
NO DELETE UNTIL 356 DAYS AFTER INSERT
HASHING USING "SHA2_512" VERSION "V1"
PARTITION BY RANGE(transaction_date)
(PARTITION p0 VALUES LESS THAN (TO_DATE('31.03.2023','DD.MM.YYYY'))
,PARTITION p1 VALUES LESS THAN (TO_DATE('30.06.2023','DD.MM.YYYY'))
,PARTITION p2 VALUES LESS THAN (TO_DATE('30.09.2023','DD.MM.YYYY'))
,PARTITION p3 VALUES LESS THAN (TO_DATE('31.12.2023','DD.MM.YYYY'))
);
```

Creating blockchain tables

```
select   *
from     all_tab_columns
where    table_name = 'TRANSACTION_LEDGER';
```

OWNER	TABLE_NAME	COLUMN_NAME	DATA_TYPE
HROUG2023	TRANSACTION_LEC	TRANSACTION_ID	NUMBER
HROUG2023	TRANSACTION_LEC	TRANSACTION_DAT	DATE
HROUG2023	TRANSACTION_LEC	TRANSACTION_USE	VARCHAR2

Hidden columns

Column Name	Data Type	Description
ORABCTAB_INST_ID$	NUMBER (22)	Instance ID of the database instance into which the row is inserted.
ORABCTAB_CHAIN_ID$	NUMBER (22)	Chain ID of the chain, in the database instance, into which the row is inserted. Valid values for chain ID are 0 through 31.
ORABCTAB_SEQ_NUM$	NUMBER(22)	Sequence number of the row on the chain. Each row inserted into a chain of a blockchain table is assigned a unique sequence number that starts with 1. The sequence number of a row is 1 higher than the sequence number of the previous row in the chain. Missing rows can be detected using this column. The combination of instance ID, chain ID, and sequence number uniquely identifies a row in the blockchain table.
ORABCTAB_CREATION_TIME$	TIMESTAMP WITH TIME ZONE	Time, in UTC format, when a row is created.
ORABCTAB_USER_NUMBER$	NUMBER (22)	User ID of the database user who inserted the row.
ORABCTAB_HASH$	RAW(2000)	Hash value of the row. The hash value is computed based on the row content of the row and the hash value of the previous row in the chain.
ORABCTAB_SIGNATURE$	RAW(2000)	User signature of the row. The signature is computed using the hash value of the row.
ORABCTAB_SIGNATURE_ALG$	NUMBER(22)	Signature algorithm used to produce the user signature of a signed row.
ORABCTAB_SIGNATURE_CERT$	RAW(16)	GUID of the certificate associated with the signature on a signed row.
ORABCTAB_SPARE$	RAW(2000)	This column is reserved for future use.

Each row in blockchain table contains hidden columns that are populated by the database when inserted row is commited

Altering blockchain tables

- It is only possible to increase retention period of table and rows:

```
alter table transaction_ledger no drop until 16 days idle;

alter table transaction_ledger_partitioned
no delete until 712 days after insert locked;
```

Viewing blockchain tables

```
select   *
from     user_blockchain_tables;
```

TABLE_NAME	ROW_RETENTION	ROW_RETENTION_LOCKED	TABLE_INACTIVITY_RETENTION
TRANSACTION_LEDGER	356	NO	16
TRANSACTION_LEDGER_PARTITIONED	712	YES	16

Inserting into blockchain tables

```
insert into transaction_ledger
(transaction_id
,transaction_date
,transaction_user)
values
(1
,sysdate
,'test_user'
);
```

Querying data from blockchain tables

```
select   *
from     transaction_ledger;
```

TRANSACTION_ID	TRANSACTION_DATE	TRANSACTION_USER
1	8/10/2023, 8:09:22 AM	test_user

Querying data from blockchain tables

```
select  transaction_id
,       ORABCTAB_INST_ID$
,       ORABCTAB_CHAIN_ID$
,       ORABCTAB_SEQ_NUM$
,       ORABCTAB_CREATION_TIME$
,       ORABCTAB_USER_NUMBER$
,       ORABCTAB_HASH$
,       ORABCTAB_SIGNATURE$                    commit;
,       ORABCTAB_SIGNATURE_ALG$
,       ORABCTAB_SIGNATURE_CERT$
,       ORABCTAB_SPARE$
from    transaction_ledger;
```

TRANSACTION_ID	ORABCTAB_INST_ID$	ORABCTAB_CHAIN_ID$	ORABCTAB_SEQ_NUM$	ORABCTAB_CREATION_TIME$	ORABCTAB_USER_NUMBER$	ORABCTAB_HASH$	ORABCTAB_SIGNATURE$	ORABCTAB_SIGNATURE_ALG$	ORABCTAB_SIGNATURE_CERT$	ORABCTAB_SPARE$
1	1	23	1	08.08.23 11:08:09,721036000 GMT	187	B0A02F4DDF497...	(null)	(null)	(null)	(null)
2	1	23	2	08.08.23 11:10:35,548158000 GMT	187	C255AFA2EFF35...	(null)	(null)	(null)	(null)
3	1	23	3	08.08.23 11:10:35,548674000 GMT	187	1C0B467ACC3F9...	(null)	(null)	(null)	(null)
4	1	23	4	08.08.23 11:10:35,549194000 GMT	187	0281823D0983F...	(null)	(null)	(null)	(null)

Deleting data from blockchain tables

- Only rows outside retention period can be deleted:

```
delete   transaction_ledger
where    transaction_id = 4;
```

SQL Error: ORA-05715: operation not allowed on the blockchain or immutable table
05715. 0000 - "operation not allowed on the blockchain or immutable table"
*Cause: The table was insert-only table and, therefore, could not be updated or deleted.

Deleting data from blockchain tables

```
DECLARE
    l_num_rows NUMBER;
BEGIN
   DBMS_BLOCKCHAIN_TABLE.DELETE_EXPIRED_ROWS
      (schema_name            => 'HROUG2023'
      ,table_name             => 'TRANSACTION_LEDGER'
      ,before_timestamp       => TO_DATE('31.03.2023','DD.MM.YYYY')
      ,number_of_rows_deleted => l_num_rows);

   DBMS_OUTPUT.PUT_LINE('Number of rows deleted = ' || l_num_rows);
END;
```

PL/SQL procedure successfully completed.
Number of rows deleted = 0

Dropping blockchain tables

- Can be dropped if no rows or wasn't modified within retention period
- Must be within user schema or user must have DROP ANY TABLE privilege
- Use Purge option

```
drop table transaction_ledger_partitioned purge;
```

Blockchain table can be dropped if it has no rows or it wasn't modified within retention period
Must be within user schema or user must have DROP ANY TABLE privilege
It is recommended to use Purge option when dropping blockchain table

Adding certificate

- X.509 digital certificate
- Add certificate to database as BLOB
- Stored certificate ID - sign and verifiy signed blockchain table row
- Multiple certificates
- Row can have only one signature

Obtain an X.509 digital certificate from a Certificate Authority (CA).
Add certificate to database as BLOB
Use stored certificate ID to sign and verifiy signed blockchain table row
Multiple certificates can be used to sign a row but each row can have only one signature

Adding certificate

- Creating signing key:

```
openssl genrsa -out bc_signing_key.pem 2048
```

- Creating self-signing certificate:

```
openssl req -new -x509 -outform pem -sha512 -days 3650 \
-nodes \
-out bc_signing_certificate.pem \
-key bc_signing_key.pem \
-subj
"/C=HR/ST=Zagreb/L=Somwhere/O=HROUG2023/OU=IN2/CN=Tomislav
/emailAddress=tomislavku@in2.hr"
```

Adding certificate

```
DECLARE
    file        BFILE;
    buffer      BLOB;
    amount      NUMBER := 32767;
    cert_id  RAW(16);
BEGIN
    file := BFILENAME('BC_CERT_DIR', 'bc_signing_certificate.pem');
    DBMS_LOB.FILEOPEN(file);
    DBMS_LOB.READ(file, amount, 1, buffer);
    DBMS_LOB.FILECLOSE(file);
    DBMS_USER_CERTS.ADD_CERTIFICATE(buffer, cert_id);
    DBMS_OUTPUT.PUT_LINE('Certificate ID = ' || cert_id);
END;
```

```
Certificate GUID = 02A55DA59D470E5EE0630100007FF6E2
```

Adding certificate

- Query certificates from data dictionary views:
  - DBA_CERTIFICATES
  - CDB_CERTIFICATES
  - USER_CERTIFICATES

CERTIFICATE_ID	USER_NAME	DISTINGUISHED_NAME	CERTIFICATE
02A55DA59D470E5EE0630100007FF6E2	HROUG2023	EMAIL=tomislavku@in2.hr,	(BLOB)

We can query informations about existing certificates from data dictionary views

Deleting certificate

```
declare
    certificate_guid RAW(16):=02A73D7872B912B0E0630100007F3E27';
begin
    DBMS_USER_CERTS.DROP_CERTIFICATE(certificate_guid);
end;
```

Adding signature to blockchain table row

- Signing a row is optional
- Additional security against tampering
- Oracle database verifies:
  - that the current user owns the row being updated
  - that the hash provided matches stored hash value of the row.
- Digital certificate signs blockchain table row
- Supported signature algorithms:
  - SIGN_ALGO_RSA_SHA2_256
  - SIGN_ALGO_RSA_SHA2_384
  - SIGN_ALGO_RSA_SHA2_512

Signing a row with user signature is optional
Provides additional security against tampering
Oracle database verifies:
that the current user owns the row being updated
that the hash provided matches stored hash value of the row.
Digital certificate is used when adding a signature to a blockchain table row
Supported signature algorithms:
SIGN_ALGO_RSA_SHA2_256
SIGN_ALGO_RSA_SHA2_384
SIGN_ALGO_RSA_SHA2_512

Adding signature to blockchain table row

- To add a signature to a blockchain table row:
  - existing signature of that row must be NULL
  - INSERT privilege on the blockchain table is needed

Adding signature to blockchain table row

```
select transaction_id
, ORABCTAB_INST_ID$
, ORABCTAB_CHAIN_ID$
, ORABCTAB_SEQ_NUM$
, ORABCTAB_SIGNATURE$
, ORABCTAB_SIGNATURE_ALG$
, ORABCTAB_SIGNATURE_CERT$
, ORABCTAB_SPARE$
from transaction_ledger;
```

TRANSACTION_ID	ORABCTAB_INST_	ORABCTAB_CHAI	ORABCTAB_SEQ_	ORABCTAB_SIGNATURE$	ORABCTAB_SIGNATURE_ALG$	ORABCTAB_SIGNATURE_CERT$	ORABCTAB_SPARE$
1	1	31	1	(null)	(null)	(null)	(null)
3	1	31	2	(null)	(null)	(null)	(null)
2	1	25	1	(null)	(null)	(null)	(null)
4	1	25	2	(null)	(null)	(null)	(null)

Adding signature to blockchain table row

```
declare
    l_row_data blob;
    l_buffer raw(4000);
    l_inst_id binary_integer;
    l_chain_id binary_integer;
    l_seq_num binary_integer;
    l_row_len binary_integer;
    l_file utl_file.file_type;
begin
select orabctab_inst_id$,orabctab_chain_id$,orabctab_seq_num$
into l_inst_id,l_chain_id,l_seq_num
from transaction_ledger where transaction_id = 2;
dbms_blockchain_table.get_bytes_for_row_signature(schema_name => 'HROUG2023'
                                                 ,table_name  => 'TRANSACTION_LEDGER'
                                                 ,instance_id => l_inst_id
                                                 ,chain_id    => l_chain_id
                                                 ,sequence_id => l_seq_num
                                                 ,data_format => 1
                                                 ,row_data    => l_row_data);
l_row_len := dbms_lob.getlength(l_row_data);
dbms_lob.read(l_row_data, l_row_len, 1, l_buffer);
l_file := utl_file.fopen('BC_CERT_DIR','transaction2.dat','wb', 32767);
utl_file.put_raw(l_file, l_buffer, true);
utl_file.fclose(l_file);
end;
```

Adding signature to blockchain table row

Signing row digest:

```
openssl dgst -sha512 \
        -sign bc_signing_key.pem \
        -out transaction2.sha512 \
        transaction2.dat
```

Adding signature to blockchain table row

```
DECLARE
l_inst_id binary_integer;
l_chain_id binary_integer;
l_sequence_no binary_integer;
l_file BFILE;
l_source_off integer := 1;
l_destination_off integer := 1;
l_signature blob;
l_cert_guid RAW (16) := HEXTORAW('02A55DA59D470E5EE0630100007FF6E2');
BEGIN
select orabctab_inst_id$,orabctab_chain_id$,orabctab_seq_num$
into l_inst_id,l_chain_id,l_sequence_no
from transaction_ledger where transaction_id = 2;
l_file := bfilename('BC_CERT_DIR', 'transaction2.sha512');
dbms_lob.createtemporary(l_signature, false);
dbms_lob.fileopen(l_file);
dbms_lob.loadblobfromfile(l_signature,l_file,dbms_lob.getlength(l_file),l_
estination_off,l_source_off);
dbms_lob.fileclose(l_file);
```

Adding signature to blockchain table row

```
dbms_blockchain_table.sign_row(schema_name => 'HROUG2023'
,table_name        => 'TRANSACTION_LEDGER'
,instance_id       => l_inst_id
,chain_id          => l_chain_id
,sequence_id       => l_sequence_no
,hash              => NULL
,signature         => l_signature
,certificate_guid  => l_cert_guid
,signature_algo    => DBMS_BLOCKCHAIN_TABLE.SIGN_ALGO_RSA_SHA2_512);
END;
```

Adding signature to blockchain table row

TRANSACTION_ID	ORABCTAB_INST_	ORABCTAB_CHAI	ORABCTAB_SEQ_	ORABCTAB_SIGNATURE$	ORABCTAB_SIGNATURE_ALG$	ORABCTAB_SIGNATURE_CERT$	ORABCTAB_SPARE$
1	1	31	1	(null)	(null)	(null)	(null)
3	1	31	2	(null)	(null)	(null)	(null)
2	1	25	1	0EA24D566588AE2D206956EC	3	028EEC4E44D00A8CE0630100007FE0AF	(null)
4	1	25	2	(null)	(null)	(null)	(null)

Generating signed digest for blockchain table

- Signed digest - metadata + last row data in each chain of a table at certain point in time
- Signature based on contents of the signed digest
- Signature using private key and certificate
- Signature and signed digest stored in repository

Signed digest consist of metadata and data about the last row in each chain of a table at certain point in time
Signature is based on contents of the signed digest
Signature is using private key and certificate of the blockchain table owner
Signature and signed digest generated at various times should be stored in repository

Generating signed digest for blockchain table

- Prerequisites:
  - certificate added to database
  - PKI private key and certificate of blockchain table owner stored in a wallet:
    - For PDB: WALLET_ROOT/pdb_guid/bctable/
    - For non-CDB: WALLET_ROOT/bctable/

Prerequisites:

certificate of blockchain table owner must be added to database

PKI private key and certificate of blockchain table owner must be stored in a wallet:

For PDB: WALLET_ROOT/pdb_guid/bctable/

For non-CDB: WALLET_ROOT/bctable/

Generating signed digest for blockchain table

Set WALLET_ROOT (as CDB SYSDBA):

```
ALTER SYSTEM SET WALLET_ROOT =
'/opt/oracle/product/23c/dbhomeFree/admin/FREE' SCOPE=SPFILE;
```

Create folder structure under WALLET_ROOT:

```
SELECT pdb_name, guid FROM dba_pdbs;
```

PDB GUID: F87259FB7D3C3519E0530100007F5D4C

```
cd /opt/oracle/product/23c/dbhomeFree/admin/FREE/
mkdir F87259FB7D3C3519E0530100007F5D4C/bctable
```

Generating signed digest for blockchain table

Creating wallet:

```
orapki wallet create -wallet
/opt/oracle/product/23c/dbhomeFree/admin/FREE/F87259FB7D3C3519E0530100007
F5D4C/bctable/ -auto_login_only
```

Package signed cerificate and key:

```
openssl pkcs12 -export -in bc_signing_certificate.pem \
-inkey bc_signing_key.pem \
-out private_key_bct_owner.p12 -passout pass:"oracle"
```

Generating signed digest for blockchain table

Adding key to wallet:

```
orapki wallet import_pkcs12 \
-wallet
/opt/oracle/product/23c/dbhomeFree/admin/FREE/F87259FB7D3C3519E053010000
7F5D4C/bctable/ \
-auto_login_only -pkcs12file
/home/oracle/my_wallet/private_key_bct_owner.p12 \
-pkcs12pwd "oracle"
```

Generating signed digest for blockchain table

```
DECLARE
l_signed_bytes BLOB;
l_signed_row_array SYS.ORABCTAB_ROW_ARRAY_T;
l_certificate_guid RAW(2000) := '02A55DA59D470E5EE0630100007FF6E2';
l_signature RAW(2000);
BEGIN
dbms_lob.createtemporary(l_signed_bytes, false);
l_signature := DBMS_BLOCKCHAIN_TABLE.GET_SIGNED_BLOCKCHAIN_DIGEST
(schema_name => 'HROUG2023'
,table_name => 'TRANSACTION_LEDGER'
,signed_bytes => l_signed_bytes
,signed_rows_indexes => l_signed_row_array
,schema_certificate_guid => l_certificate_guid
,signature_algo => dbms_blockchain_table.SIGN_ALGO_RSA_SHA2_512);
insert into bc_signed_digests (creation_date,signed_digest,signed_bytes)
values(sysdate,to_blob(l_signature),l_signed_bytes);

DBMS_OUTPUT.PUT_LINE('Certificate GUID = ' || l_certificate_guid);
DBMS_OUTPUT.PUT_LINE('Signature length = ' || UTL_RAW.LENGTH(l_signature));
DBMS_OUTPUT.PUT_LINE('Number of chains = ' || l_signed_row_array.count);
DBMS_OUTPUT.PUT_LINE('Signature content buffer length = ' ||
DBMS_LOB.GETLENGTH(l_signed_bytes));
END;
```

Verifying the Integrity of Blockchain Tables

1. DBMS_BLOCKCHAIN_TABLE.VERIFY_ROWS - verify links
2. DBMS_BLOCKCHAIN_TABLE.GET_SIGNED_BLOCKCHAIN_DIGEST at time T1
3. DBMS_BLOCKCHAIN_TABLE.GET_SIGNED_BLOCKCHAIN_DIGEST at time T2
4. DBMS_BLOCKCHAIN_TABLE.VERIFY_TABLE_BLOCKCHAIN - verify integrity between T1 and T2

Repeat steps 2 through 4 at different time periods

1. Verify the links between all the chains in the blockchain table using DBMS_BLOCKCHAIN_TABLE.VERIFY_ROWS procedure.

   If row contains a user signature, the row signature is also verified
2. Generate a signature and signed digest for the blockchain table using DBMS_BLOCKCHAIN_TABLE.GET_SIGN

ED_BLOCKCHAIN_DIGEST function at time T1.

Generated details and generation date and time should be stored in repository that must be outside the database that stores the blockchain table, e.g. another relational database

2. Generate a signature and signed digest for the blockchain table using DBMS_BLOCKCHAIN_TABLE.GET_SIGNED_BLOCKCHAIN_DIGEST function at time T2. Generated details and generation date and time should be stored in repository.
3. Verify integrity of rows that were created between T1 and T2 using DBMS_BLOCKCHAIN_TABLE.VERIFY_TABLE_BLOCKCHAIN procedure. Inputs for this procedure are signed digests generated at T1 and T2.

Steps 2 through 4 should be repeated at different time periods, to verify the integrity of rows inserted between different time periods.

Verifying Rows

```
DECLARE
      l_rows_verified NUMBER;
BEGIN
   DBMS_BLOCKCHAIN_TABLE.VERIFY_ROWS
   (schema_name => 'HROUG2023'
   ,table_name => 'TRANSACTION_LEDGER'
   ,low_timestamp => NULL
   ,high_timestamp => NULL
   ,instance_id => 1
   ,chain_id => NULL
   ,number_of_rows_verified => l_rows_verified
   ,verify_signature => TRUE);
   dbms_output.put_line('Number of rows verified in instance id 1 = ' ||
   l_rows_verified);
END;

Number of rows verified in instance id 1 = 5
```

Verifying Integrity of Rows

```
DECLARE
l_signature RAW(2000);
l_signed_row_array SYS.ORABCTAB_ROW_ARRAY_T;
l_signed_bytes1 BLOB;
l_certificate_guid RAW(2000) := '02A55DA59D470E5EE0630100007FF6E2';
l_signed_bytes2 BLOB;
l_rows_verified NUMBER;
BEGIN
SELECT signed_bytes INTO l_signed_bytes1 FROM bc_signed_digests WHERE trunc(creation_date)
= trunc(SYSDATE-1);

l_signature := DBMS_BLOCKCHAIN_TABLE.GET_SIGNED_BLOCKCHAIN_DIGEST
(schema_name => 'HROUG2023'
,table_name => 'TRANSACTION_LEDGER'
,signed_bytes => l_signed_bytes2
,signed_rows_indexes => l_signed_row_array
,schema_certificate_guid => l_certificate_guid
,signature_algo => dbms_blockchain_table.SIGN_ALGO_RSA_SHA2_512);
```

Verifying Integrity of Rows

```
DBMS_BLOCKCHAIN_TABLE.VERIFY_TABLE_BLOCKCHAIN
(signed_bytes_latest       => l_signed_bytes2
,signed_bytes_previous     => l_signed_bytes1
,number_of_rows_verified   => l_rows_verified);

dbms_output.put_line('Rows verified = ' || l_rows_verified);
END;

Rows verified = 5
```

If you want to learn more

- https://docs.oracle.com/en/database/oracle/oracle-database/23/admin/managing-tables.html#GUID-E7151628-AF04-48D4-9CB4-F72417AFC391
- https://docs.oracle.com/en/database/oracle/oracle-database/23/arpls/dbms_blockchain_table.html#GUID-8B000001-AE8B-42EA-8BF3-E590BCBA6657
- https://apexapps.oracle.com/pls/apex/r/dbpm/livelabs/view-workshop?wid=875&p180_gb_clicked=Y&session=105569203428298

If you want to learn more

- https://www.oracle.com/blockchain/#blockchain-platform-tab
- https://ec.europa.eu/digital-building-blocks/wikis/display/EBSI/Home
- https://www.hyperledger.org/
- https://101blockchains.com/enterprise-blockchain-framework/

hroug'23
godišnja konferencija