<Intro>
<Cardano Smart Contract>
<PLUTUS>

Plutus là gì?

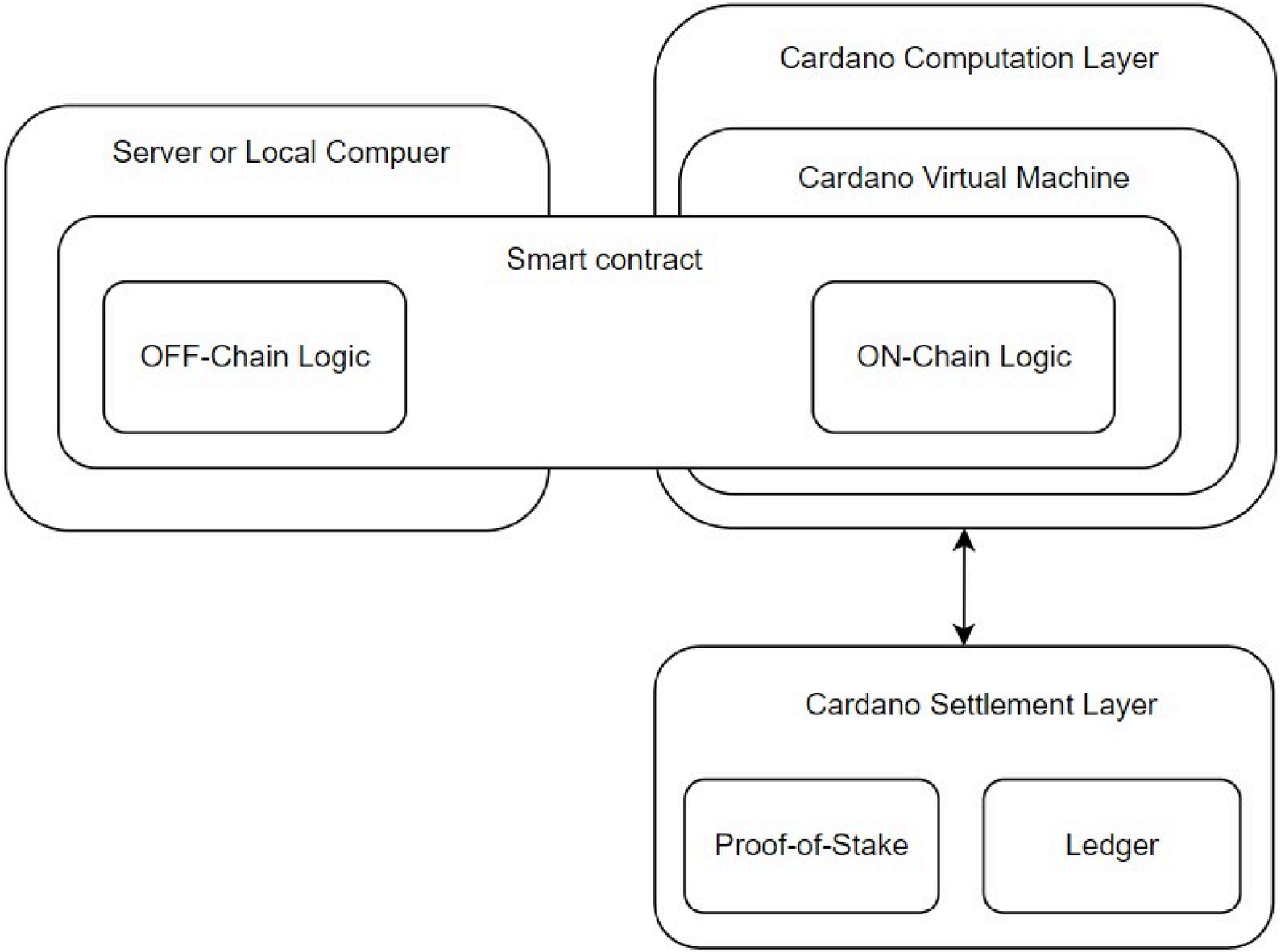
- Tổng quan về Nền tảng Plutus, PlutusTx và Plutus Core.

Trình Xác Thực, UPLC, CBOR

- Hiểu về Untyped Plutus Core (UPLC) và quá trình tuần tự hóa qua CBOR.

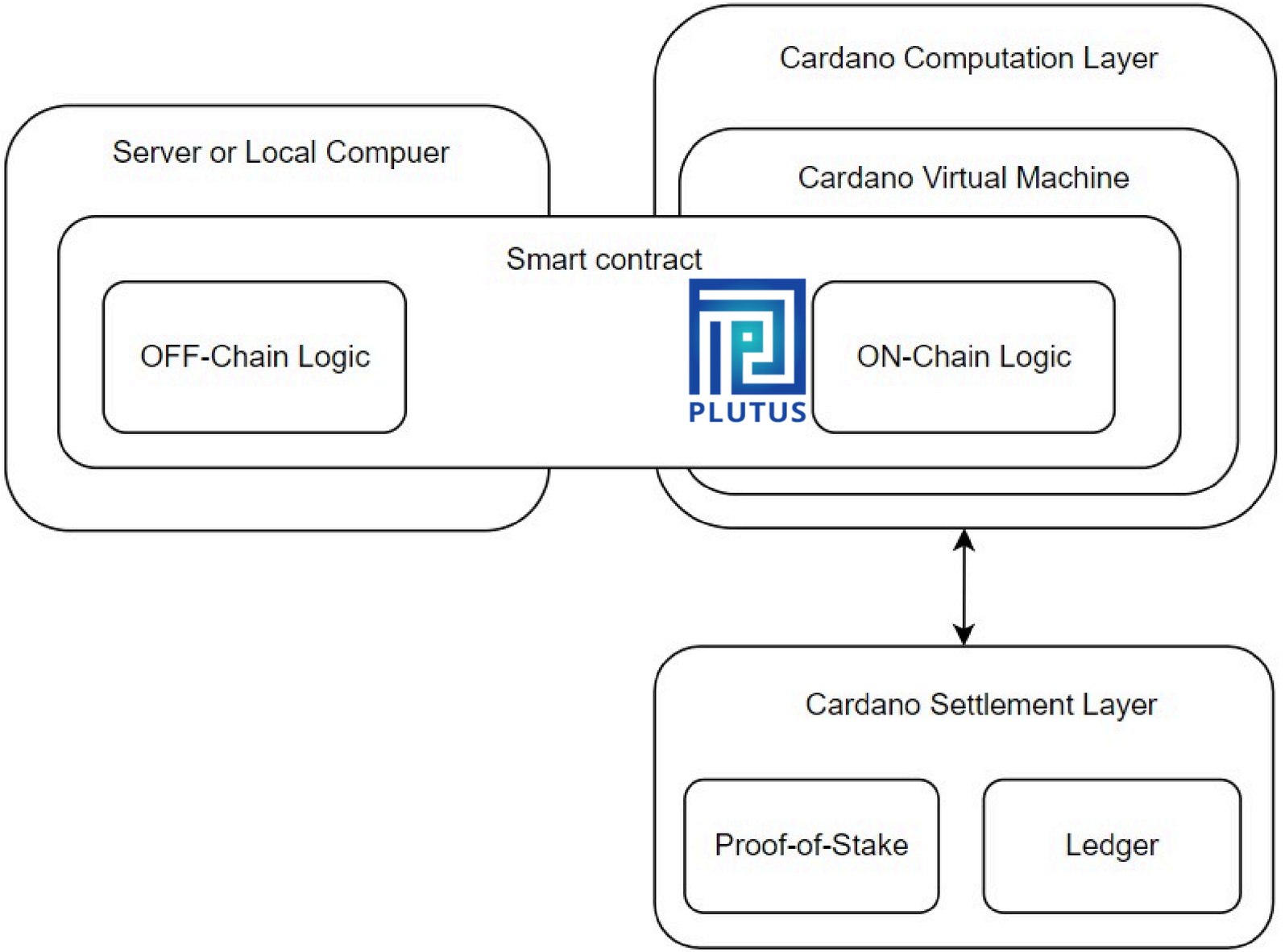
Băm và Chữ Ký Số

- Vai trò của mật mã trong việc bảo mật hợp đồng thông minh.



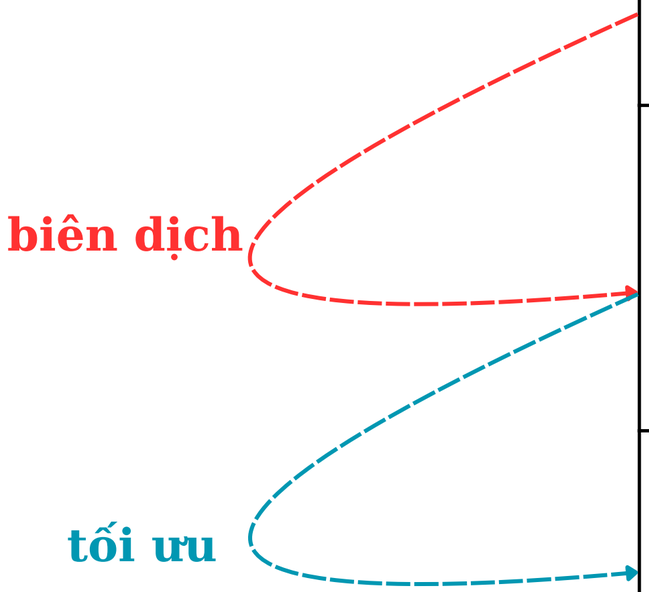
Thực thi SC

Xử lí giao dịch



Thực thi SC

Xử lí giao dịch



Thành phần	Vai trò	Đặc điểm chính
Plutus	Nền tảng phát triển hợp đồng thông minh	Bao gồm công cụ, môi trường và API
PlutusTx	Ngôn ngữ viết hợp đồng thông minh	Subset của Haskell, biên dịch thành Plutus Core
Plutus Core	Ngôn ngữ cấp thấp để thực thi	Biên dịch từ PlutusTx, tối ưu cho blockchain
UPLC	Bytecode thực thi cuối cùng trên blockchain	Loại bỏ kiểu dữ liệu, tối ưu hiệu suất

Cấu trúc của UPLC

- Lệnh Lambda (Lam): Biểu diễn hàm và tham số.
- Lệnh Ứng Dụng (App): Thực thi hàm với đối số.
- Lệnh Biến Số (Var): Đại diện cho các biến.
- Lệnh Hằng Số (Const): Đại diện cho các giá trị hằng số như số nguyên, chuỗi.
- Lệnh Kết Hợp (Builtin): Các hàm dựng sẵn như phép cộng, trừ, băm dữ liệu.

```
program
  { "version": "1.0" }
  (lam x (lam y (addInteger x y)))
```

Thành phần

Plutus 4

PlutusTx 1

Plutus Core 2

UPLC 3

Giảm kiểu dữ liệu phức tạp và cấu trúc cao cấp

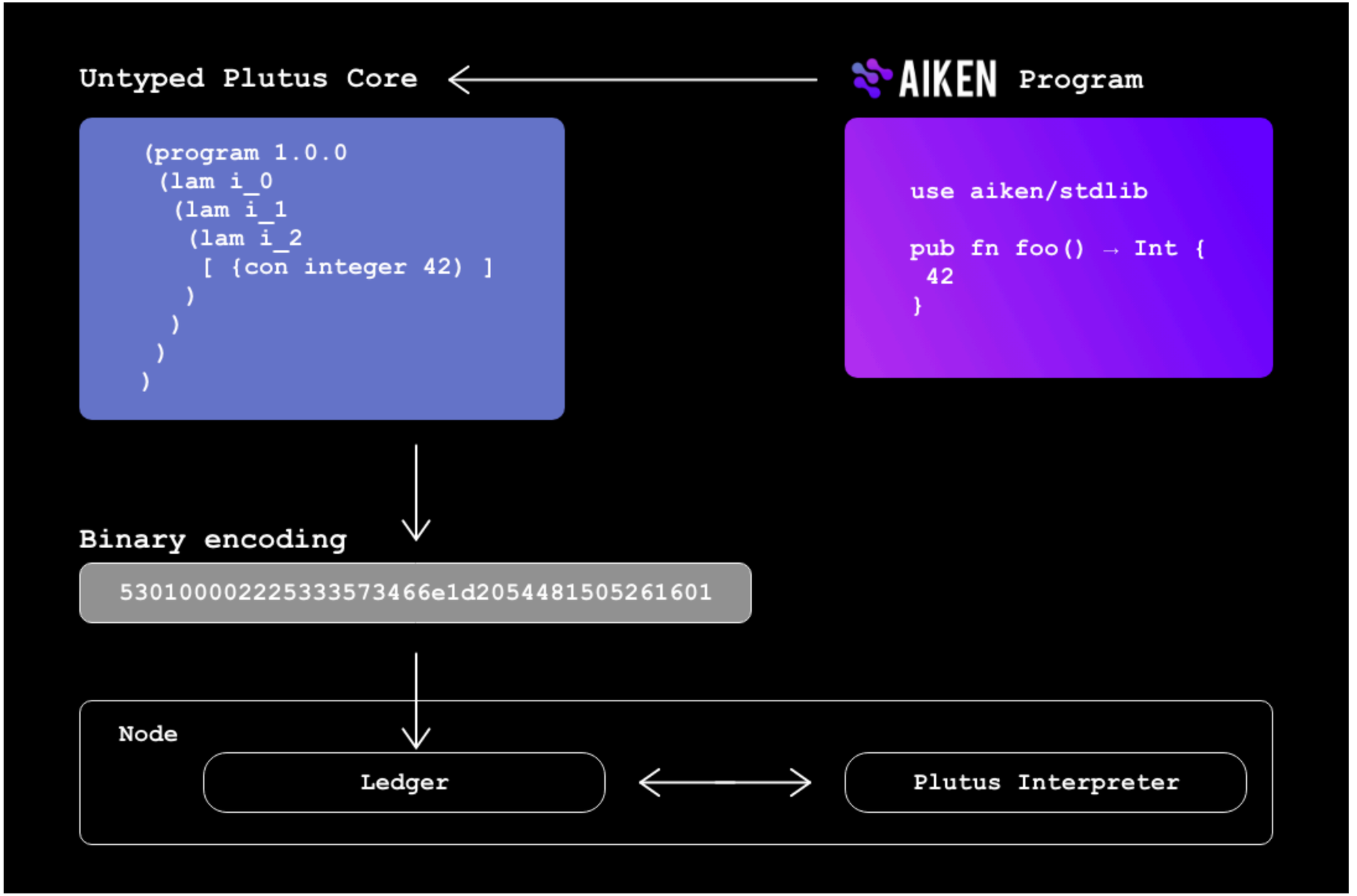
loại bỏ hệ thống kiểu

```
mkValidator :: () -> Integer -> ScriptContext -> Bool
mkValidator _ redeemer _ = redeemer > 10

validator :: Validator
validator = mkValidatorScript $$(compile [|| mkValidator ||])
```

```
(program 1.0.0
  (lam redeemer
    (ifThenElse
      (greaterThan redeemer 10)
      true
      false
    )
  )
)
```

```
(program 1.0.0
  [
    [ (lam redeemer
        [
          [ (builtin greaterThan) redeemer 10 ]
          True
          False
        ]
      )
    ]
  ]
```


Untyped Plutus Core
AIKEN Program
(program 1.0.0
(lam i_0
(lam i_1
(lam i_2
[(con integer 42)]
)
)
)
)
use aiken/stdlib
pub fn foo() → Int {
42
}
Binary encoding
5301000022253335734 66e1d2054481505261601
Node
Ledger
Plutus Interpreter

1. Tạo giao dịch

```
cardano-cli transaction build \
  --alonzo-era \
  --tx-in <TX-IN> \
  --tx-out <RECIPIENT>+<AMOUNT> \
  --change-address <CHANGE-ADDRESS> \
  --witness-override <NUM-WITNESSES> \
  --protocol-params-file protocol.json \
  --out-file tx.raw
```

2. Tạo script kiểm tra tính hợp lệ

```
cardano-cli transaction build \
  --alonzo-era \
  --tx-in <TX-IN> \
  --tx-in-script-file <PLUTUS_SCRIPT.plutus> \
  --tx-in-datum-file <DATUM.json> \
  --tx-in-redeemer-file <REDEEMER.json> \
  --tx-in-collateral <COLLATERAL-TX-IN> \
  --tx-out <RECIPIENT>+<AMOUNT> \
  --change-address <CHANGE-ADDRESS> \
  --protocol-params-file protocol.json \
  --out-file tx.raw
```

3. Ký giao dịch

```
cardano-cli transaction sign \
  --tx-file tx.raw \
  --signing-key-file <SIGNING-KEY.skey> \
  --mainnet | --testnet-magic <MAGIC-NUMBER> \
  --out-file tx.signed
```

4. Gửi giao dịch

```
cardano-cli transaction submit \
  --tx-file tx.signed \
  --mainnet | --testnet-magic <MAGIC-NUMBER>
```

```
simple-validator/
│
├── src/
│   └── SimpleValidator.hs      # Haskell code for the Validator script
│
├── scripts/
│   ├── validator.plutus        # Compiled Plutus Core script
│   ├── datum.json              # JSON file for Datum
│   ├── redeemer.json           # JSON file for Redeemer
│   ├── protocol.json           # Protocol parameters
│   └── tx.raw                  # Raw transaction file
│
├── keys/
│   ├── payment.skey            # Signing key (private key)
│   ├── payment.vkey            # Verification key (public key)
│   └── collateral.tx           # UTXO for collateral
│
└── build.sh                    # Script to automate compilation and transaction creation
```

```haskell
{-# INLINABLE mkValidator #-}
mkValidator :: () -> Integer -> ScriptContext -> Bool
mkValidator _ redeemer _ = redeemer > 10
```

```
(program 1.0.0
  (lam redeemer
    (ifThenElse
      (greaterThanInteger redeemer 10)
      true
      false
    )
  )
)
```

```
4d01000033222220051200120011223344556677889aabbccddeeff001122334455
```

Khái Niệm	Định Nghĩa	Vai Trò	Ví Dụ
Validator	Hàm kiểm tra điều kiện giao dịch	Xác thực giao dịch trên chuỗi	Redeemer > 10
UPLC	Mã trung gian đã biên dịch	Thực thi mã trên blockchain	(ifThenElse ...)
CBOR	Định dạng lưu trữ nhị phân	Giảm chi phí lưu trữ & truyền	"cborHex": "4d0100..."

Thành Phần	Mô Tả	Vai Trò
Datum (Dữ Liệu)	Trạng thái lưu trữ trong UTXO	Cung cấp thông tin liên quan đến giao dịch
Redeemer (Dữ Liệu Chứng Minh)	Giá trị người dùng cung cấp khi chi tiêu UTXO	Đối số chính cho logic của Validator
Script Context (Ngữ Cảnh Script)	Thông tin về giao dịch hiện tại	Chứa các đầu vào, đầu ra và thông tin liên quan đến UTXO

Tình Huống Giả Định

- Chúng ta muốn tạo một hợp đồng thông minh để khóa một số tiền trong UTXO.
- Khi người dùng muốn chi tiêu UTXO đó, họ phải cung cấp một Redeemer (giá trị chứng minh).
- Điều kiện để giao dịch hợp lệ là:

Datum + Redeemer >= 100 và giao dịch phải trả ít nhất 2 ADA.`

```haskell
-- Hàm xác thực chính
{-# INLINABLE mkValidator #-}
mkValidator :: Integer -> Integer -> ScriptContext -> Bool
mkValidator datum redeemer ctx =
    traceIfFalse "Không đủ điều kiện!" (datum + redeemer >= 100) && -- Điều kiện hợp lệ
    traceIfFalse "Không đủ 2 ADA!" hasMinimumAda                    -- Đảm bảo trả ít nhất 2 ADA
  where
    info :: TxInfo
    info = scriptContextTxInfo ctx

    hasMinimumAda :: Bool
    hasMinimumAda = any (\o -> txOutValue o >= lovelaceValueOf 2000000) (txInfoOutputs info)

-- Biên dịch Validator
validator :: Validator
validator = mkValidatorScript $$(PlutusTx.compile [|| mkValidator ||])
```

Thành Phần	Vai Trò	Trong Code	Cụ Thể
Datum	Trạng thái lưu trữ trong UTXO	datum	Số nguyên, tham số đầu tiên.
Redeemer	Dữ liệu chứng minh khi chi tiêu UTXO	redeemer	Số nguyên, tham số thứ hai.
ScriptContext	Thông tin giao dịch hiện tại	ctx	Được dùng để lấy TxInfo và kiểm tra đầu ra.

CBOR

```
{
  "type": "PlutusScriptV2",
  "description": "Validator Script",
  "cborHex": "4d01000033222220051200120011"
}
```

- **Giảm kích thước dữ liệu**: Tối ưu hóa chi phí lưu trữ và truyền tải.
- **Biểu diễn dữ liệu phức tạp**: Như scripts, transactions, và metadata.
- **Dễ dàng phân tích:** Các giá trị CBOR có thể được giải mã thành JSON hoặc các định dạng có cấu trúc khác.

CBOR

Cấu Trúc CBOR Cơ Bản

Trong CBOR, mỗi thẻ dữ liệu (header) gồm hai phần:

Major Type (3 bit đầu tiên của byte): Xác định kiểu dữ liệu.

Additional Information (5 bit còn lại): Xác định kích thước dữ liệu hoặc độ dài

Major Type	Giá Trị	Kiểu Dữ Liệu	Ý Nghĩa
0	000	Số nguyên không âm	0-23 hoặc thêm byte mở rộng.
1	001	Số nguyên âm	-1 - (-24) hoặc thêm byte mở rộng.
2	010	Chuỗi byte	Mảng byte.
3	011	Chuỗi ký tự UTF-8	Chuỗi văn bản (String).
4	100	Danh sách (Array)	Danh sách các phần tử.
5	101	Bản đồ (Map)	Cặp key-value.
6	110	Thẻ đặc biệt (Tag)	Gán ý nghĩa đặc biệt cho dữ liệu.
7	111	Giá trị đặc biệt	false, true, null, float.

CBOR

HEX

a2647479706565737472696e676568656c6c6f65776f726c64

CBOR HEX

a264747970656573747269 6e676568656c6c6f65776f726c64

Major Type 5

a2 (hex) → 1010 0010 (binary)

Chia Thành 2 Phần:
3 bit đầu tiên (101):
Đây là Major Type 5, đại diện cho Map (bản đồ).
5 bit còn lại (00010):
Đây là giá trị thêm (Additional Information) → xác định kích thước của Map.
Ở đây, 00010 bằng 2, tức là Map này chứa 2 cặp key-value.

CBOR HEX

Major Type 3

64 (hex) → 0110 0010 (binary)

Major Type 3 (011): Dữ liệu là chuỗi ký tự UTF-8.
Additional Information 4: Chuỗi có độ dài 4 ký tự.

--> Xét tiếp 4 ký tự

a264747970656573747269 6e676568656c6c6f65776f726c64

Major Type 5

CBOR HEX

Major Type 3

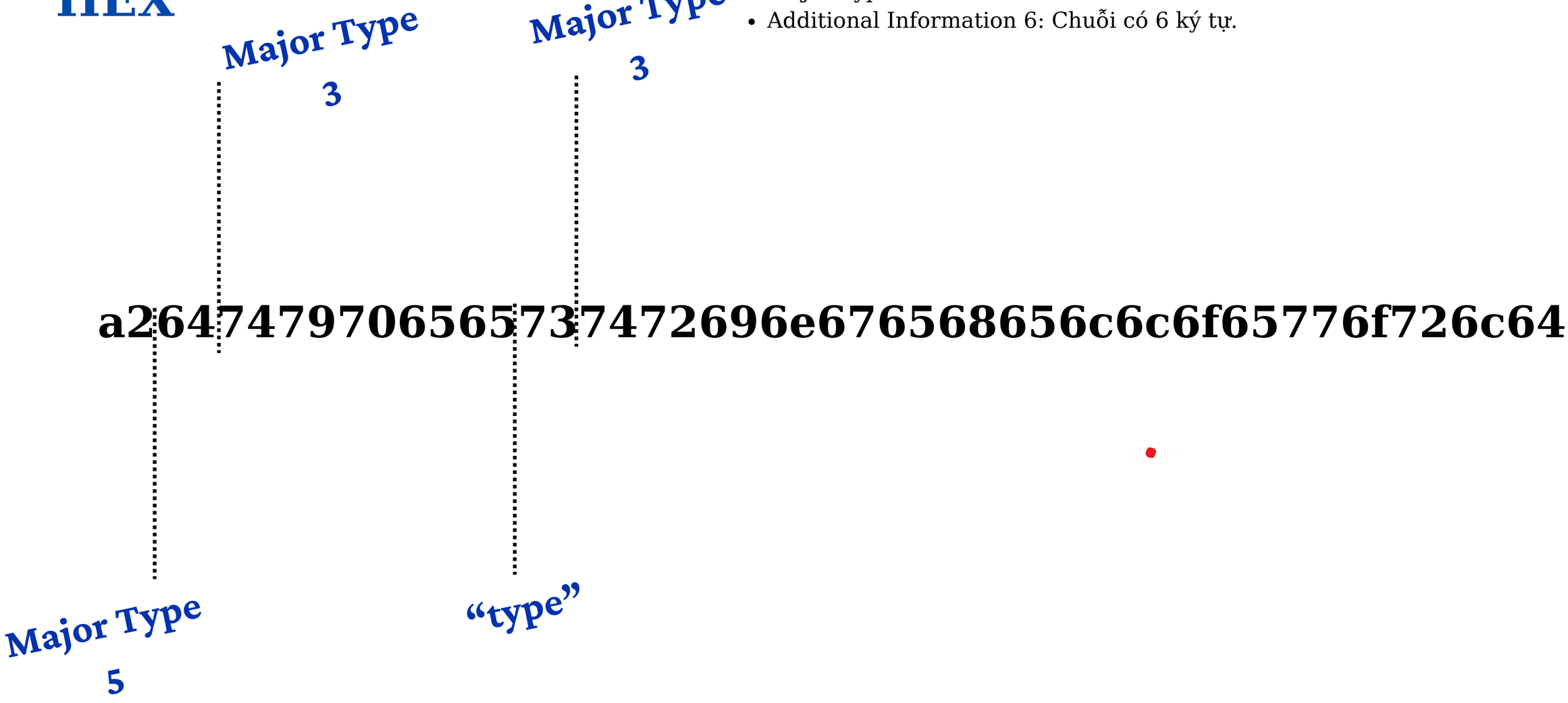
a26474797065657374726 96e676568656c6c6f65776f726c64

Theo UTF -8
74 : t
77: y
70: p
65: e

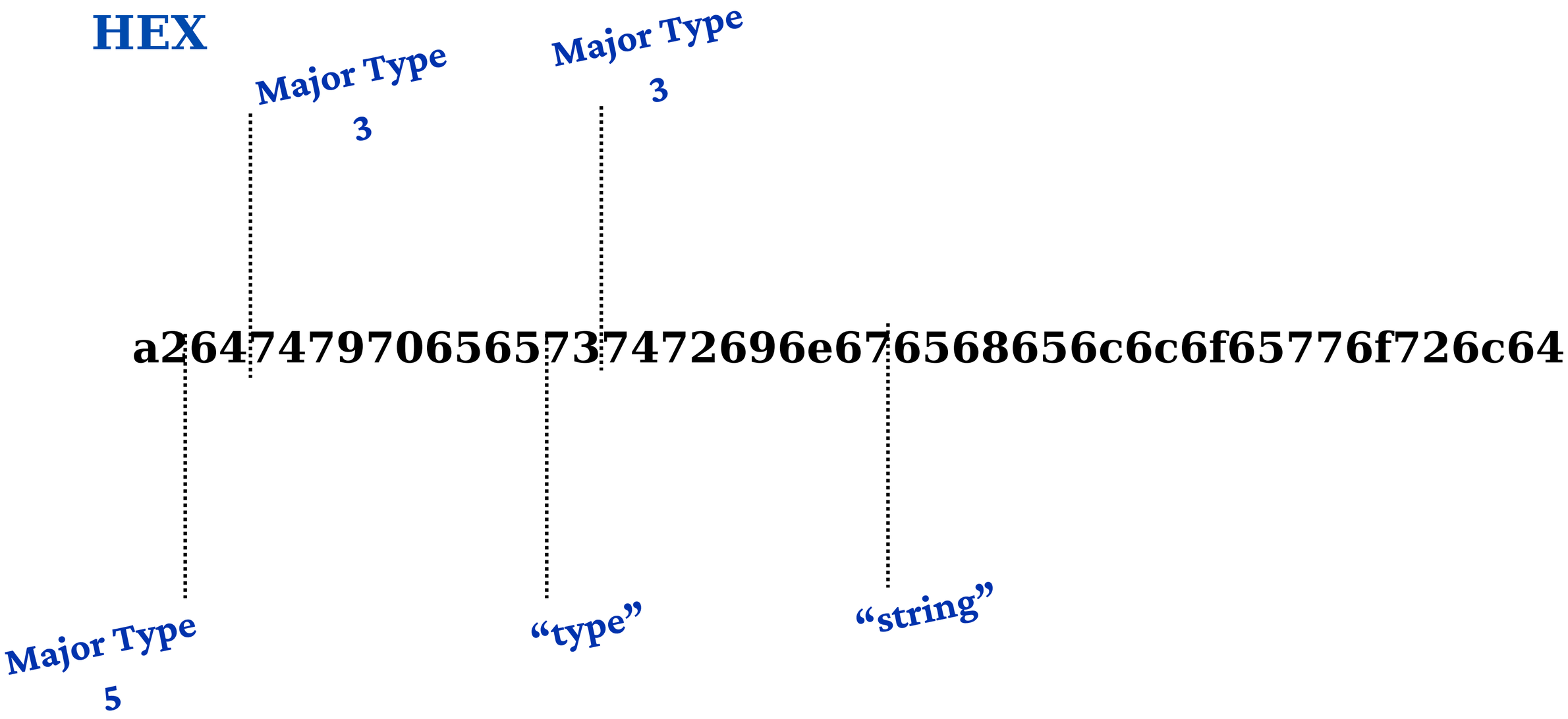
Major Type 5

CBOR HEX

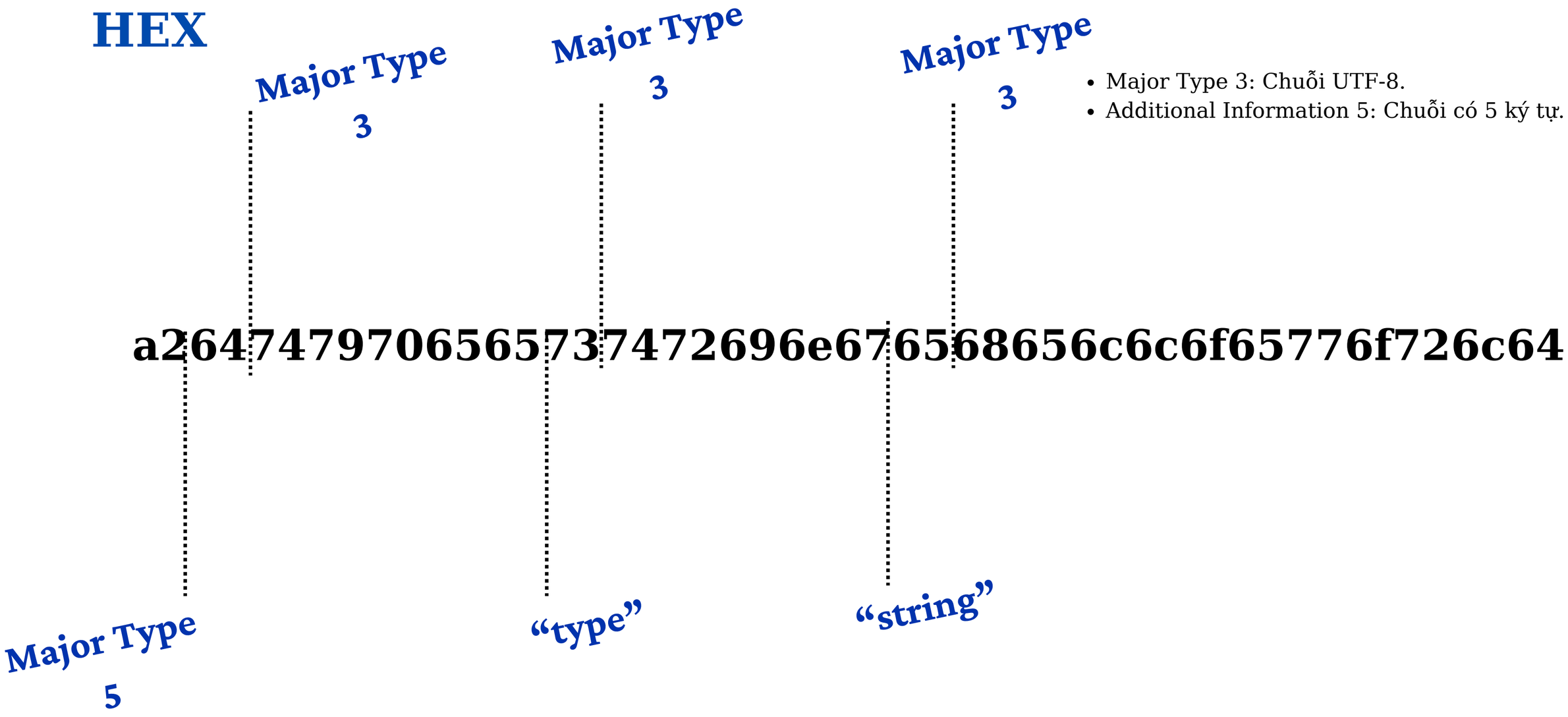
- Major Type 3: Chuỗi UTF-8.
- Additional Information 6: Chuỗi có 6 ký tự.



CBOR HEX

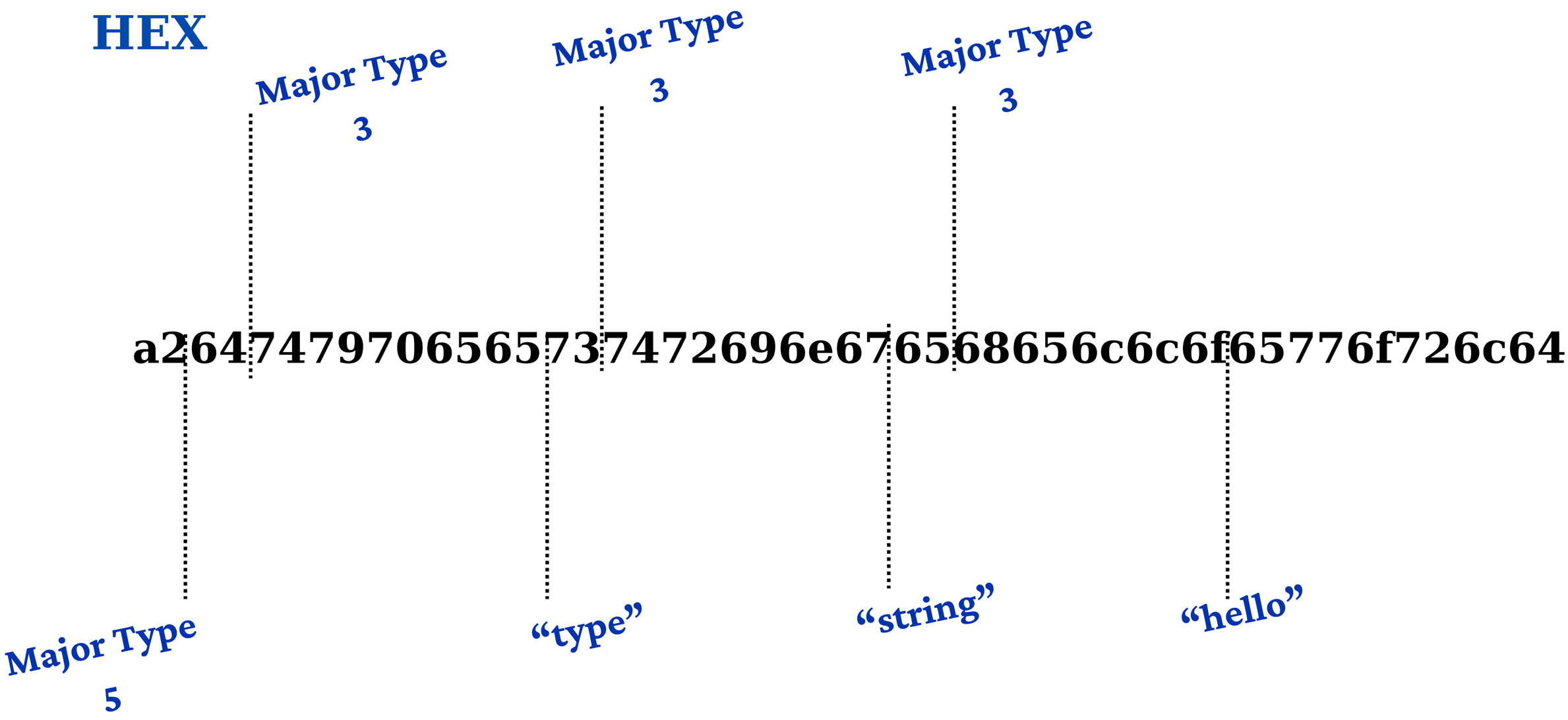

Major Type 3
Major Type 3
a264747970656573 7472696e67 6568656c6c6f65776f726c64
Major Type 5
“type”
“string”

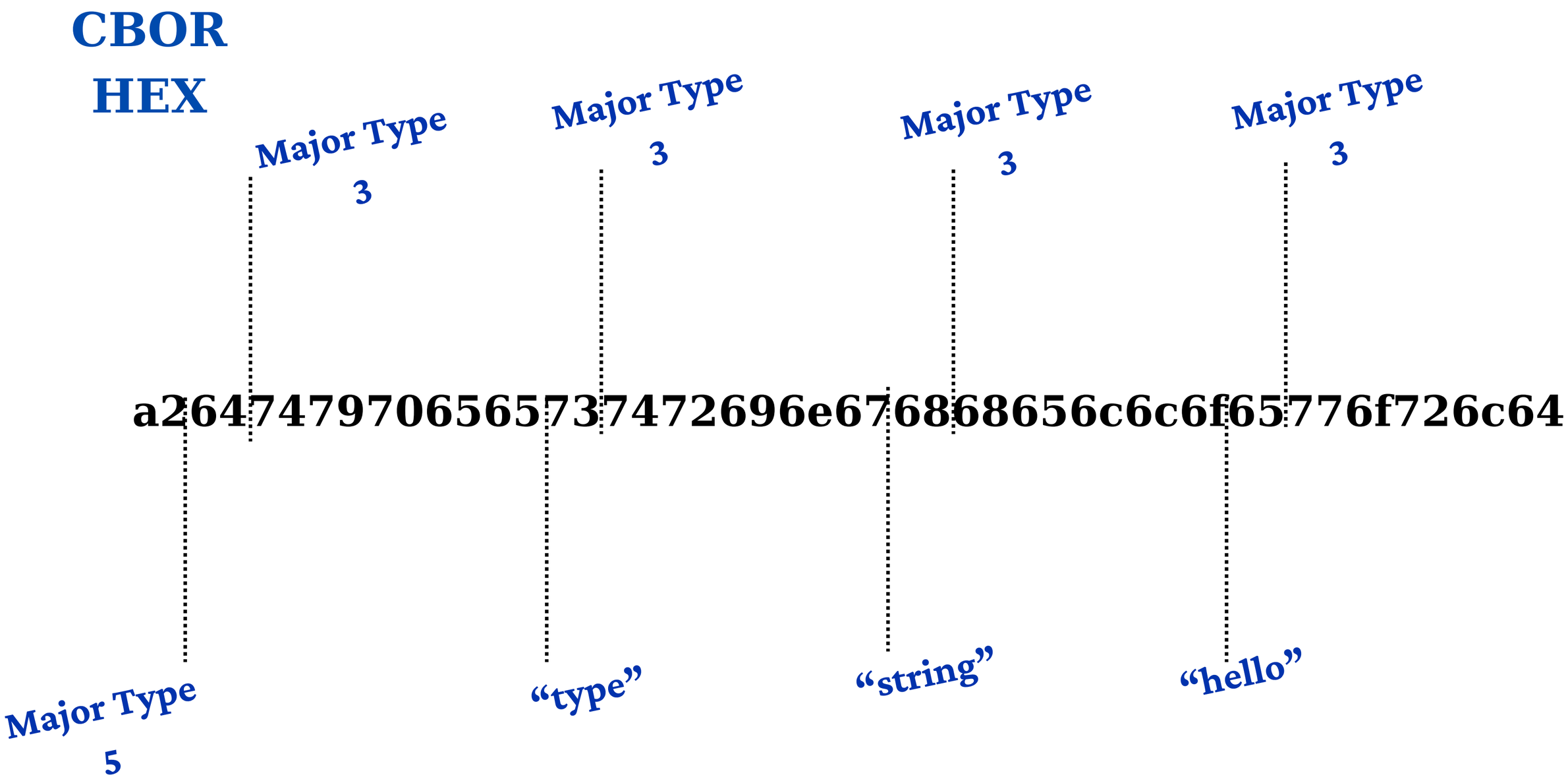
CBOR HEX



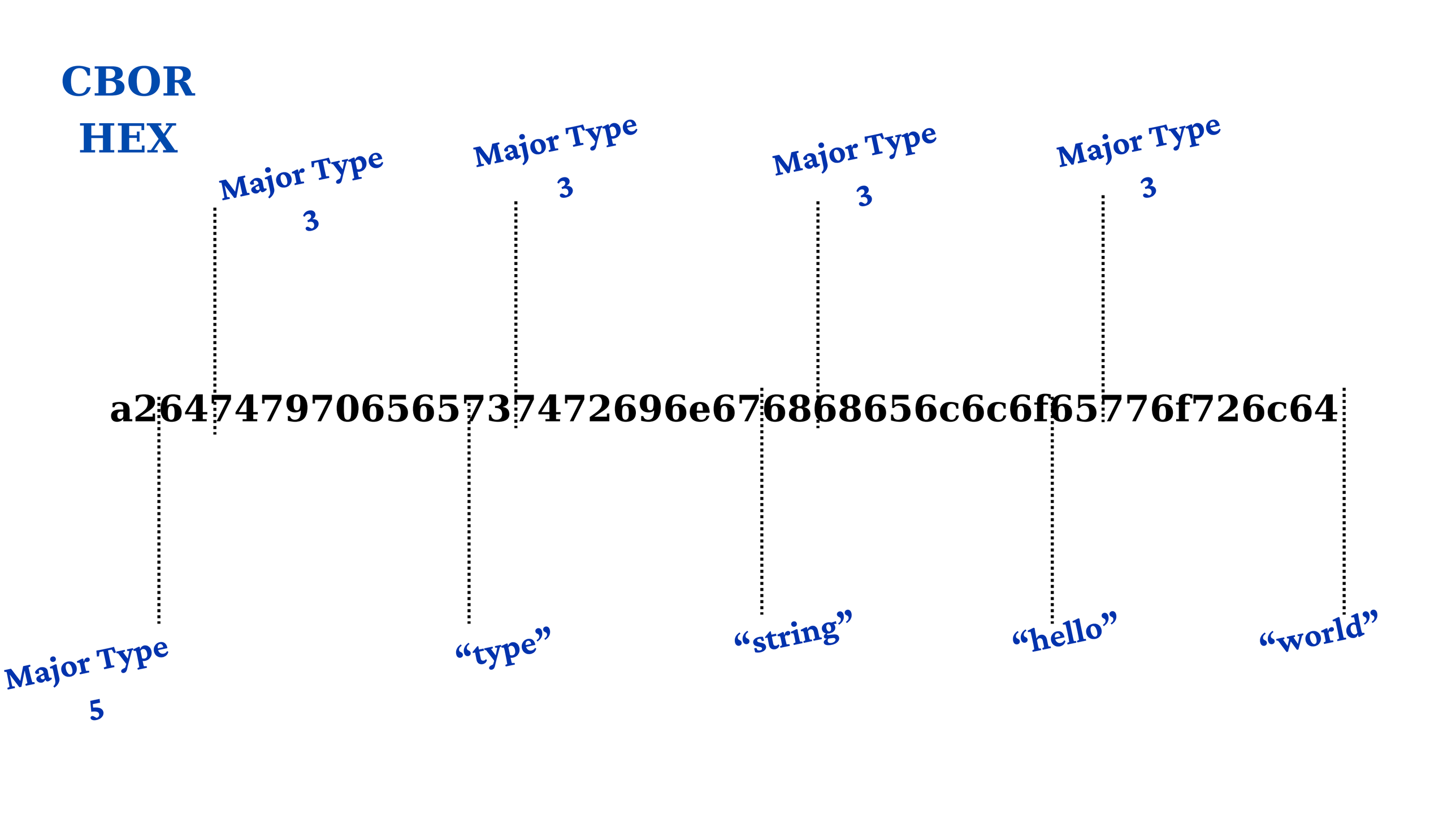
- Major Type 3: Chuỗi UTF-8.
- Additional Information 5: Chuỗi có 5 ký tự.

CBOR HEX


Major Type 3
Major Type 3
Major Type 3
a264747970656573747269 6e67656 8656c6c6f65776f726c64
Major Type 5
“type”
“string”
“hello”



- Major Type 3: Chuỗi UTF-8.
- Additional Information 5: Chuỗi có 5 ký tự.


CBOR
HEX
Major Type
5
Major Type
3
Major Type
3
Major Type
3
Major Type
3
a264747970656573747269e676868656c6c6f65776f726c64
“type”
“string”
“hello”
“world”

a2647479706565737472696e676568656c6c6f65776f726c64

Các Thành Phần Chính Được CBOR Hóa Trên Cardano

1. **Transaction Body (thân giao dịch):**
   - Bao gồm đầu vào (inputs), đầu ra (outputs), fee và thời gian.
2. **Transaction Witnesses (chữ ký và scripts):**
   - Mã hóa chữ ký số, khóa công khai, và script xác thực.
3. **Transaction Metadata:**
   - Các thông tin bổ sung do người dùng tùy chỉnh.
4. **Plutus Scripts và UPLC:**
   - Scripts được tuần tự hóa thành CBOR để xác thực giao dịch.

Chuẩn bị JSON

```
{
  "inputs": [
    {"txId": "abc123", "index": 0}
  ],
  "outputs": [
    {"address": "addr1...", "value": 1000000}
  ],
  "fee": 200000
}
```

Xác định Major Type & Độ dài

Mã hóa từng phần

Kết hợp thành phần

```
A3                  # Map có 3 key-value
   6A 696E70757473  # "inputs"
      81            # Array có 1 phần tử
         A2         # Map 2 cặp key-value (txId và index)
            66 74784964  # "txId"
            43 ABC123    # Giá trị txId
            65 696E646578 # "index"
            00           # Giá trị 0
   6B 6F757470757473  # "outputs"
      81            # Array có 1 phần tử
         A2         # Map 2 cặp key-value
            67 6164647231 # "address"
            18 64         # "value"
   63 666565      # "fee"
      18 2E       # Giá trị 200000
```

```
A3 6A 696E70757473 81 A2 66 74784964 43 ABC123 65 696E646578 00 6B 6F757470757473 81 A2 67 6164647231 18 64 63 666565 18 2E
```

```
cardano-cli transaction build --tx-in abc123#0 \
  --tx-out addr1...+1000000 --fee 200000 \
  --out-file tx.raw

cardano-cli transaction view --tx-file tx.raw
```

Tạo và kiểm tra CBOR (đang được binary encoded)

```
cardano-cli transaction view --tx-file tx.raw --out-file tx.cbor
xxd tx.cbor
```

Để xem nội dung CBOR dưới dạng HEX hoặc JSON,
giải mã file bằng lệnh xxd hoặc cardano-cli transaction view.

2. CDDL Là Gì?

CDDL (Concise Data Definition Language) là một ngôn ngữ định nghĩa dữ liệu nhỏ gọn, được thiết kế để mô tả cách các đối tượng dữ liệu được tổ chức và tuần tự hóa. Nó thường được sử dụng để xác định các cấu trúc CBOR trên Cardano.

Chức Năng CDDL:

Mô tả cấu trúc dữ liệu: Xác định các loại dữ liệu như chuỗi, số nguyên, danh sách, và bản đồ.
Tính Tương Thích: Dễ dàng diễn giải và giải mã bởi các hệ thống khác nhau.
Định Dạng Chuẩn: Dựa trên RFC 8610.

- CDDL xác định cấu trúc dữ liệu trên Cardano nhưng không đảm bảo thứ tự tuần tự hóa duy nhất.
- Các nhà phát triển nên lưu trữ byte gốc và xử lý linh hoạt các đối tượng đã giải mã để tránh sai lệch khi tính hash.
- Đây là yếu tố quan trọng khi triển khai hợp đồng thông minh và giao dịch trên Cardano.

Định nghĩa theo CDDL

```
transaction = {
  "inputs": [* input],    ; Danh sách đầu vào giao dịch
  "outputs": [* output], ; Danh sách đầu ra giao dịch
  "fee": uint,            ; Phí giao dịch, kiểu số nguyên không âm
  ? "metadata": tstr      ; Metadata tùy chọn (chuỗi văn bản)
}

input = {
  "txId": tstr,           ; ID giao dịch đầu vào
  "index": uint           ; Vị trí trong UTXO
}

output = {
  "address": tstr,        ; Địa chỉ người nhận
  "value": uint           ; Giá trị (số lovelace) gửi đi
}
```

Tuân theo CDDL

```
{
  "inputs": [
    {"txId": "abc123", "index": 0}
  ],
  "outputs": [
    {"address": "addr1xyz...", "value": 1000000}
  ],
  "fee": 200000,
  "metadata": "Transaction Metadata Example"
}
```

- Tuân theo CDDL giúp đảm bảo dữ liệu hợp lệ trên Cardano, giảm rủi ro sai lệch và tăng tính tương thích.
- Tuy nhiên, cần cẩn trọng với thứ tự tuần tự hóa và các trường tùy chọn để tránh thay đổi giá trị hash.
- Giải pháp tốt nhất là lưu trữ byte gốc và áp dụng chuẩn hóa tuần tự hóa như CIP-0021 để đảm bảo tính nhất quán.

==> Tóm lại: CDDL định nghĩa cấu trúc dữ liệu, nhưng developer cần xử lý cẩn thận khi serialize/deserialize để tránh sai lệch trong các giao dịch và hợp đồng thông minh.

CDDL files

Era	CDDL Specification
Byron	byron.cddl
Shelley	shelley.cddl
Allegra	allegra.cddl
Mary	mary.cddl
Alonzo	alonzo.cddl
Babbage	babbage.cddl
Conway	conway.cddl

Vai trò
Mật mã học

Chức Năng	Vai Trò Của Mật Mã	Ví Dụ
Xác thực danh tính	Đảm bảo người thực hiện hành động là chủ sở hữu hợp lệ	Chữ ký số (txSignedBy trong Plutus)
Bảo vệ tính toàn vẹn	Ngăn chặn dữ liệu bị thay đổi trong quá trình thực thi	Sử dụng hàm băm SHA-256
Tính không thể phủ nhận	Người ký không thể từ chối đã thực hiện hành động	Chữ ký số trong giao dịch blockchain
Bảo mật giao tiếp	Bảo vệ dữ liệu khi gửi và nhận	Mã hóa bằng mật mã bất đối xứng
Hợp đồng đa chữ ký	Đảm bảo nhiều bên xác nhận trước khi thực thi hợp đồng	Multi-signature trong smart contracts

Khái Niệm	Vai Trò	Ví Dụ	Dạng Giá Trị
Hashing	Bảo đảm tính toàn vẹn và nhận dạng dữ liệu	Băm dữ liệu giao dịch hoặc UTXO	Chuỗi Hexadecimal (SHA-256)
Chữ ký số	Xác minh danh tính, tính toàn vẹn và tính không thể phủ nhận	Ký giao dịch và xác minh trên blockchain	Chuỗi Hexadecimal hoặc Dãy Byte

Sử dụng
HASHING

Trường Hợp	Mục Đích	Ví Dụ
Quản lý UTXO	Tạo TxID và nhận dạng đầu ra giao dịch	3b24d5a1f8790...
Địa chỉ ví	Băm khóa công khai để tạo địa chỉ ví	addr1qxy3w49...
Validator Script	So sánh giá trị băm trong logic hợp đồng	sha2_256 datum == expectedHash
Metadata (NFT)	Đảm bảo tính toàn vẹn của siêu dữ liệu	Hash file: 3b24d5a1f8790...
Stake Pools	Tạo pool ID và xác thực danh tính operator	pool1g8s8kjr9z...
Block và Blockchain	Liên kết block và đảm bảo toàn vẹn chuỗi	8eab0f18c9c3b...

Thuộc Tính	Độ Dài	Giải Thích
Hash thông thường	32 byte (256 bit)	Độ dài chuẩn của các hàm băm trên Cardano.
Hash của Credentials	28 byte (224 bit)	Bao gồm khóa (keys) hoặc script được dùng làm thông tin xác thực.
Policy ID	28 byte (224 bit)	Là giá trị băm của một tagged script (script có gắn thẻ).
Hash của Verification Key	28 byte (224 bit)	Giá trị băm của khóa xác minh (verification

Hầu hết các hàm băm trên Cardano có độ dài 32 byte, ngoại trừ credentials (khóa hoặc script) và policy ID, có độ dài 28 byte để tối ưu hóa.

Sử dụng
CHỮ KÝ SỐ

Trường Hợp	Mục Đích	Vai Trò Chữ Ký Số
Quản lý giao dịch	Xác minh người gửi và bảo vệ dữ liệu	Ký giao dịch bằng khóa riêng tư
Staking & Delegation	Ủy quyền stake và nhận thưởng	Ký giao dịch ủy quyền
Plutus Smart Contracts	Xác minh quyền thực hiện hành động	txSignedBy kiểm tra chữ ký
Tạo địa chỉ ví	Bảo mật và ẩn danh địa chỉ ví	Băm khóa công khai và kết hợp chữ ký
Multi-Signature Transactions	Giao dịch yêu cầu nhiều chữ ký xác minh	Xác minh nhiều chữ ký bằng khóa công khai
Stake Pool Registration	Đăng ký và xác minh danh tính stake pool	Ký thông tin pool bằng khóa riêng tư

Tính Năng	Plutus (Cardano)	Solidity (Ethereum)
Ngôn ngữ cơ sở	Haskell	JavaScript-like
Tính bảo mật	Cao, nhờ hệ thống kiểu chặt chẽ	Trung bình, dễ xảy ra lỗi
Tính xác minh toán học	Có thể chứng minh hợp đồng	Hạn chế
Kiến trúc	Tách biệt On-chain và Off-chain	Gộp chung trong một môi trường
Tính tất định	Có	Khó đoán kết quả

```haskell
{-# INLINABLE mkValidator #-}
mkValidator :: BuiltinData -> BuiltinData -> ScriptContext -> Bool
mkValidator _ redeemer _ =
    let r = (unsafeFromBuiltinData redeemer :: Integer)
    in traceIfFalse "Redeemer không lớn hơn 10!" (r > 10)

validator :: Validator
validator = mkValidatorScript $$(PlutusTx.compile [|| mkValidator ||])
```

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract SimpleValidator {
    function validate(uint256 input) public pure returns (bool) {
        require(input > 10, "Input không lớn hơn 10!");
        return true;
    }
}
```

Logic	Plutus	Solidity
Kiểm tra giá trị > 10	datum + redeemer > 10	require(input > 10, "Input không lớn hơn 10!");
Kiểu dữ liệu	Kiểm tra kiểu chặt chẽ (Integer)	Dùng uint256, dễ thiếu sót kiểm tra số âm
Xử lý lỗi	traceIfFalse cung cấp thông tin lỗi rõ ràng	require chỉ dừng và báo lỗi ngắn gọn
Bảo mật	Rất cao, không xảy ra lỗi tràn số	Có thể bị lỗi overflow/underflow
Chi phí giao dịch	Thấp nhờ tính toán Off-chain	Cao vì tính toán diễn ra On-chain

What I wish I knew when exploring Cardano

Chủ Đề	Giải Thích Ngắn Gọn	Lưu Ý Quan Trọng
CBOR / CDDL	CBOR là định dạng nhị phân như JSON, CDDL mô tả cấu trúc CBOR.	Dùng để tuần tự hóa dữ liệu.
Địa chỉ Byron	Địa chỉ cũ trước Shelley, không được khuyến khích.	Không tương thích với Plutus.
Khoảng thời gian hợp lệ	Giới hạn thời gian giao dịch hợp lệ bằng "validity intervals".	Giảm thiểu các lỗi xác thực.
Chiến lược tuần tự hóa	Không có chuẩn duy nhất, nên bảo toàn byte gốc khi giải mã.	Tránh lỗi khi tái tuần tự hóa.
Hàm băm (Hash Digests)	Sử dụng blake2b 256 bit cho băm thông thường, 224 bit cho khóa.	Băm đúng chuẩn CDDL.
Phần thưởng & Rút tiền	Phần thưởng lưu trong tài khoản đặc biệt và cần giao dịch để rút.	Rút bằng chứng chỉ stake.
Scripts gốc (Native Scripts)	Ngôn ngữ scripting đơn giản trước khi có Plutus.	Dùng cho địa chỉ multisig.
Thủ thuật 'clean'	Nút có thể xác thực lại toàn bộ chuỗi để kiểm tra tính toàn vẹn.	Tránh lỗi khi khối bị phát lại.

Các CIP Liên Quan Đến Lập Trình Plutus:

- **CIP-30:** Wallet API, tiêu chuẩn giao tiếp giữa ví và ứng dụng phi tập trung (DApp).
- **CIP-68:** Định dạng token gốc mở rộng, giúp lưu trữ metadata tùy chỉnh.
- **CIP-31:** Đầu vào tham chiếu (Reference Inputs), giúp các script truy cập dữ liệu on-chain mà không cần chi tiêu UTXO.
- **CIP-32:** Dữ liệu tập trung (Inline Datums), giúp lưu trữ dữ liệu trực tiếp trên UTXO.
- **CIP-33:** Script tập trung (Reference Scripts), giảm chi phí lưu trữ bằng cách tái sử dụng script.