CoinCar : Système décentralisé de location de voitures entre particuliers fondé sur la blockchain BitCoin

---

IF-4-SERE

Use case

CoinCar allows individuals to rent out their cars (a sort of AirBnB for cars, or OuiCar) or other vehicles.

We assume that the cars are smart connected cars. Each car is equipped with a long range communication module (for example, 3G/4G) as well as a short range communication module (for example, Bluetooth, NFC). This implies that the car can communicate over the Internet as well as with user devices (such as smartphones) located in the vicinity.

The cars are electronically accessible by users through their smart phones. A car grants access only to a user who is authorized.

At any given time, a car has one owner, and can have one or no renter. The owner of the car can authorize one (and only one) renter at a time to access and use the car for a fixed duration. The owner of the car can also transfer ownership of the car, that is, sell the car to another user.

The owner of a car can publish the availability of the car for some chosen dates, as well as the rent/unit of time, address, and other details of the car (brand, model, etc.).

Renters can search for available cars that correspond to their preferred criteria. They can specify the desired location, rent, type, reputation (see below), etc.

When a renter discovers an available car that he wishes to rent, he sends a request to the owner for a specific duration along with mandatory credentials, which include national identity card (or equivalent document), driving license, and insurance. The owner can then approve the request by granting access to the renter for the requested duration.

The renter is able to access the car during the authorized duration. However, a condition for access is that the renter pays in advance for the entire duration. The car is locked down after the duration expires. Moreover, the car monitors its location and limits the mobility of the car as defined and published by the owner (e.g., no trip abroad).

The system also manages the reputation of the cars as well as the renters. After each car rental transaction, the renter can leave a rating and a comment about the car (for example, about its cleanliness, performance, etc.) and the owner of the car can do the same about the renter (for example, if he returned the car in a good condition). The feedback by the renter can be given in an open manner (that is, the owner knows the identity of the renter) or it can be given in a private (“anonymous”) manner (that is, the owner does not learn the identity of the renter). The same applies for the feedback given by the owner. Note: we will not consider the inference of a feedback provider’s identity in this TD.

The purpose of the TD is to study and design a solution for implementing the CoinCar service on top of the Bitcoin blockchain.

Exercise 1 (Identity Management) (Bitcoin Address)

There is no central authority to issue or manage identities on the Bitcoin blockchain.
Describe how identities are generated for users (owners, renters, car manufacturers) such that they are uniquely identified on the blockchain.
Calculate the probability that two users will both generate the same identity (address collision) if there are 7 billion users in total (approximately the population of Earth). Note: There are 2^160 possible Bitcoin addresses.

>>> Generate a pair of keys: private/public keys. The private key is used to sign data. Transactions use the hash of the public key (called "*address*" in Bitcoin jargon) to identify the user. More precisely: first, the user himself generates an elliptic curve private/public key pair; then he hashes the public key (using a combination of SHA256 and RIPEMD160 hash functions) => output = 160 bit long hash. This hash is finally converted in Base58Check format (some other formats are allowed). The output of this conversion is the actual *address* of the user.

Collision probability: 3 possible interpretations:

- If one considers a specific public address (and the corresponding private key) owned by a specific user, the probability (or risk) that another specific user owns the same private key is 1 / (2^160)
- If one considers a specific public address (and the corresponding private key) owned by a specific user, the probability (or risk) that another user (whoever she is) owns the same private key is 1- ((2^160-1)/ (2^160))^7.10^9. Proof: there are 2^160-1 addresses different from the address under consideration (let's call it A). The probability that a user creates an address different from address A is therefore (2^160-1) / (2^160). Consequently, the probability that all users create addresses different from A is ((2^160-1) / (2^160))^7.10^9. So the probability that at least one user creates for herself the address A is the opposite of the previous probability i.e., 1- (2^160-1)/ (2^160))^7.10^9.
- Now consider the set of all the users; the probability that two users own the same private key is homomorphic to the birthday paradox (aka birthday problem). See Wikipedia "Birthday Attack" article: one can expect to roll two equal addresses after approx. 1.25 sqrt(2^160) = 1.82x10^48 address computations, which is much more than the number of inhabitants on Earth

Exercise 2 (car ownership) (smart property, asset management, colored coins)

Describe how cars can be represented on the blockchain. We assume that a car is assigned a universally unique serial number by the manufacturer.
Write a protocol for a car manufacturer to transfer the ownership of a new car to the first owner.
Write a protocol for an owner of the car to sell the car to another person.
Write a protocol for an owner to prove the ownership of a car to a renter.

>>> There are several solutions for embedding data into a Bitcoin transaction:

- colored coins systems, storing the data in a variable used in an output script:
  - the classical way is to use the op-code OP_RETURN in the output script which makes the output script fail and to store the hash of the document, or the URL of a resource that store the metadata or the torrent info-hash of a

torrent file that stores the metadata (cf. colored coin COLU protocol)) in the parameter of this op-code => miners do not store it in main memory as the coin cannot be redeemed (*burnt coins*))
    - use a MULTISIG output where one or two of the public addresses is/are just (meta)data and not an actual address (this approach is used in the COLU colored coin protocol to store the SHA-256 hash of the data in order to be able to check the integrity of the data retrieved using the SHA-1 torrent hash-info of the torrent file that stores the data
    - use the hash of the data as output address (*burnt address*)
- use a separate blockchain (merge mining)
- use a separate database
- …

Note: is the data are not actually stored in the blockchain but in an external resource (incl. a torrent file), it is important to sign the data, so that the properties of immutability and non-repudiability can be maintained.

Note about OP_RETURN: (en.bitcoin.it): "OP_RETURN is a script opcode used to mark a transaction output as invalid. Since the data after OP_RETURN are irrelevant to Bitcoin payments, arbitrary data can be added into the output after an OP_RETURN". However an implicit rule is to limit the size of the data to 40 bytes. All outputs when OP_RETURN is used are unspendable and so are pruned by miners". Note: for other sites, the limit is 80 bytes (to be checked).

Regarding the exercise, one can store in the blockchain the metadata which describe the car (incl. the serial number, the official ID and all car features (brand, etc.)) using any of the previous approaches (actually, there are some limitations in data size => check the size and select an approach which fits with it).

Protocol for the car manufacturer: ownership (car_serial_nb: integrer, car_features: set of features, owner: Bitcoin_address):

1. Create new transaction using a coin with a minimal amount of satoshis (i.e., the single input of the transaction refers to an output of a previous transaction (recall (cf. course) that by design an output refers to the transaction recipient's address)); create two coins as outputs of the transaction
    - one coin is given to the owner's address (receiving a minimal amount of satoshis)
    - one coin (with a minimal amount of satoshis) is used to materialize the car ownership
        - a special marker ("ownership") is stored into the address field
        - the metadata (car_serial_number, car_features) (or the hash of the metadata or a link to the metadata: see above) is put as parameter of an OP_RETURN op-code in the script attached to this coin
    - The ownership is materialized by this transaction i.e., this transaction transfers the ownership to the buyer of the car
2. Broadcast the transaction along with the manufacturer public address and the signature of the transaction referred in the input, so that nodes can verify the manufacturer is the true owner of the input coin

Alternative: Use a colored coin protocol (for example, COLU, coinprism). The new buyer of the car sends her address to the manufacturer. The manufacturer sends a small bitcoin transaction and colors this transaction such that it represents the ownership of the car. The colored transaction includes the serial number of the car so that it is bound with the car. Coloring a transaction basically means adding some metadata (for example, the serial number of a car) to the transaction.

In both cases:

- Before delivery to the new buyer, the car is programmed by the manufacturer to recognize any owner of the coin as the owner of the car
- The ownership can be transferred by simply creating another transaction which takes as input the coin previously given to the owner; and take as output the address of the new owner. Note: the content of the "marker coin" (cf. above) can be either copied (and possibly updated) in a new marker coin or not (in this second approach, one has to track back to the first ownership transaction (created by the manufacturer) to get the metadata
- The ownership can be proven by digitally providing the public key of the owner and by signing any piece of data (e.g., the ownership transaction) with the private key of the owner (the verifier e.g., the car, can check: first, that the hash of this public key matches with the address written in the transaction output; and, second, that signature is valid) (same protocol as the one used in Bitcoin to check the ownership of an input coin). Note: before implementing this verification, one should track back the transactions from the last ownership transaction to the first ownership transaction (created by the manufacturer)

Exercise 3 (advertisement)

Describe a scheme for an owner to publish the availability of a car on the blockchain.

>>> Quite similar as previously, for instance create and broadcast a transaction with OP_RETURN op-code. The issue is how to publish the dates and the conditions of the rental. The simplest way to solve it is to store and display this information out of the blockchain, typically as a web page or a pdf document or in a torrent file e.g.:

1. Create new transaction using an input coin of minimal amount. Select as output the owner address
2. Put the URL of the availability document/site as parameter of OP_RETURN op-code in the output script of the transaction. It is wise to sign the document to prove it is genuine
3. Broadcast the transaction

Note: the car owner public key and its signature of the transaction referred in the input of the ownership transaction should be made available in some way, for instance in the advertisement document, so that renters can check the person who advertise the availability is the true owner of the car.

Note: if the content of the document is modified (e.g., after a reservation) and if the document is signed, the owner must sign the new version (for this reason, better to store the signature into the document than in the blockchain).

Alternative approach: code the availability into the OP_RETURN message (a bit tricky! Actually, not really feasible ;-)).

Exercise 4 (credential exchange)

Write a protocol for the renter to send his credentials to the owner using another channel (such as email) while leaving a verifiable trace of the credentials on the blockchain. Authorities can indeed later demand the owner to disclose the credentials and verify their integrity. How can the renter also preserve the confidentiality of the credentials from the public?

>>> For the sake of simplicity, we will assume that all credentials are collected into one digital document.

1. Renter: create a transaction with:
   - Input: a small coin possessed by the renter
   - Output: the renter address
   - OP_RETURN message:
     i. serial number of the car or (better) transaction id (i.e. hash value) of the car advertisement transaction
     ii. hash of the credential document
2. Renter: broadcast this transaction to the BTC network
3. Renter: sign the credential document using his private key and send it to the owner (this email can be encrypted with the public key of the owner) along with the id of the transaction generated in step 1
4. Owner: (decrypt the email then) hash the credential document, compare this hash value with the one stored after the OP_RETURN of the transaction generated in step 1. If the two values are equal, next step; else pb => call the renter
5. Owner: store the credential document in a safe way for the legal duration

Note: the renter may ask the owner to prove that she actually possesses the car. As noted before, this can be easily verified by providing the renter with the id of the transaction that transfer the ownership to the owner along with the owner's public key and his.her signature of the transaction (or any other data) (so as to verify that the owner possesses the private key corresponding to the public key of the output) (note: the renter can/should check that the "ownership coin" has not been transferred later to another user).

Note: the confidentiality of the credential document is guaranteed as the document is hashed. The credential document (not hashed) is not stored on the blockchain.

Exercise 5 (delegation of usage) (smart contracts)

Write a protocol for the owner of a car to delegate a temporally limited usage (renting of the car) to a user.
The renter gets access to the car only if he pays the rent in full in advance.
Discuss how such a delegation of usage could be implemented using smart contracts[1] (like the ones available in Ethereum).

[1] Cf. for instance : Smart Contracts: The Blockchain Technology That Will Replace Lawyers http://blockgeeks.com/guides/smart-contracts/ and Smart Contracts Explained: http://www.blockchaintechnologies.com/blockchain-smart-contracts.

>>>

First alternative:

1. The owner creates a transaction which transfers the ownership to the renter. This transaction has a nLockTime value equal to the date of effective rental (minus ~2 hours to let the transactions be inserted in the blockchain)
2. The renter creates a transaction which transfers the rent (payment) to the owner
3. If the owner does not see this transaction a few hours before the rental date, she can decide to cancel the rental just by creating another transaction that sends the ownership coin to himself

Second alternative: use a (trusted) third party, aka escrow:

1. The owner creates a transaction which transfers the ownership to the renter. This transaction can have a nLockTime value
2. The renter creates a transaction which transfers the rent to the owner. This transaction is actually a 2 of 3 MULTISIG transaction i.e., it requires 2 of the 3 signatures (owner, renter, escrow) to be redeemed
3. If everything goes well, after the rental, the owner and the renter sign a new transaction that transfers the money to the owner. If, for any reason, the car was not provided to the renter and if both the renter and the owner are honest, they will sign a transaction that sends back the money to the renter. In case of a dispute, the escrow decides to whom the money should be given (a lawsuit can be started if one of stakeholders (renter, owner) does not agree with the escrow's decision)

Third alternative: use a green address: green addresses refer to "trusted financial bodies" a kind of banks so that one can expect that for sure they will pay; if the renter has an "account" in such a bank it can ask it to write a transaction transferring the money to the owner (a kind of "chèque de banque" (bank check)).

Fourth alternative: create a joint transaction that takes as inputs the ownership coin + the payment coin and transfers the first to the renter and the second to the owner. This transactions needs to be signed by both the renter (for the payment coin) and the owner (for the ownership coin).

Other approach: create a second colored coin for the delegation of usage (which does not affect the ownership). Store in the metadata ("color") the conditions of usage (rental date, usage restrictions, etc.). Condition: the car should be sufficiently smart to be able to read and interpret this second type of colored coin.

Smart contracts: attach a smart contract to the transaction to transfer the ownership to the renter. The smart contract checks whether a payment of a given amount (which may be in Bitcoin) has been made by the renter to the owner. Then the smart contract transfers the ownership of the car to the renter. Issue: how to return the car to the original owner after the fixed renting duration? Not: Ethereum is much more powerful that Bitcoin for implementing smart contracts.

Back-transfer of ownership: after the rental period, the ownership should be transferred back to the owner. Furthermore, the renter should not have the capability to transfer the ownership to somebody else during the rental period! (actually, renting is not a temporary transfer of ownership but a delegation of usage). A possible solution is to use an escrow and generate a 2

of 3 MULTISIG ownership transfer transaction so that it is not possible for the renter to "sell" the car (transfer the ownership). After the car rental, the ownership is transferred back to the owner by a transaction signed by both the escrow and the owner. Note: the rental date is to be mentioned in the transaction so that the renter can check it corresponds to the deal and so that the escrow can know when it has to sign the back-transfer transaction (and send it to the owner so that this latter can add its own signature).
Another solution, as noted above, is to user a second colored coin ("rental coin") which define the conditions and duration of the rental; and which does not affect the ownership status.

Exercise 6 (driver authentication)
Write a protocol for a car to authenticate a user in order to grant access.

>>> The car reads the blockchain and finds the latest ownership transfer transaction for its (colored) coin. This transaction allows the car to determine the current owner of the car (address of the renter). The car then demands the user to digitally sign its serial number (or any other data) with his private key ("challenge") and to provide her.his public key. If the (hash of the) public key matches the current owner's address and if the digital signature is valid, the car authenticates the user and grants access.
Note: if a "rental coin" is created, a similar process should be implemented for granting access to the renters. Note: the logic of the application should decide if the owner is still allowed to access the car while the car is rented.

Exercise 7 (tax recovery)
Write an algorithm for a taxation authority to calculate the revenue earned by a car owner.

>>> If each rental payment transaction is linked to an ownership transfer or rental transaction, then the tax authority can search the blockchain for all payment transactions linked to ownership transfer or rental transactions that involve the owner, and calculate the sum all these payments.

Issue: how to link a rental payment transaction to an ownership transfer transaction? A possibility is to store the id of the ownership transaction as a "fake" address of an n of m MULTISIG (e.g., 1 of 2 if the original transaction is a "standard" transaction). Another solution is to add a "marker output" coin (i.e., a "color") and store the id of the ownership transaction in the OP_RETURN payload.

If one uses an escrow, the escrow can do the job for the tax administration.

Of course, if there is no direct link established between the owner and the rental payments, the task of the tax authority becomes hard… exactly as in real life!

Exercise 8 (alternative blockchain)
Discuss the pros and the cons of using the Bitcoin blockchain.
Identify and discuss alternative solutions based on public blockchains.
Identify issues to consider (…and propose solutions!) if one decides to implement the CoinCar service on top of a private blockchain.

>>> (cf. course) Some pros of Bitcoin

- No trusted third party
- No single point of failure
- Reliability and immutability (very high replication factor)
- (eventual) Consistency
- Non-repudiability
- Existing infrastructure
- Open infrastructure

Some cons of Bitcoin

- Security (in the ecosystem more than in the Bitcoin protocol itself)
- Total transparency; no real anonymity
- Low latency, low throughput, poor transactional scalability (cf. huge computing power for only 12140 transactions per day!)
- Waste of resources and energy due to a huge replication factor and the solving of useless puzzles
- Designed to store financial transactions, not complex data structures nor codes. Bypassing this limit requires using programming tricks (e.g., packing data into addresses, use multi-signatures to store data, nLockTime features, etc.) or using more advanced systems based on Bitcoin (e.g., colored coins systems)

Alternative public blockchain solutions

- Blockchain implementing smart contract, e.g., Ethereum, indeed make the programmer's life easier when this latter has to implement some kind of "contracts" i.e., pieces of code that are trigerred by some "events" (e.g., a payment)
- Some Blockchains(e.g. zcash, dash, monero…) have built-in privacy preserving capabilities which is a nice feature ☺
- Some blockchain uses proof-of-stake (nxt, synereo…). In a PoS sapproach, the node creator of the next block is chosen in a deterministic (or pseudo-random) way => no more puzzle to solve => less energy use

Private and consortium blockchain pros and cons (some points below can be seen either as pro or con, depending on the application context)

- "Closed" and well-known environment
- Trust is possible
- Much higher speed and transaction bandwidth
- Privacy
- Versatility (easy (easier) to change the rules of operation)
- Needs a management authority
- Smaller redundancy factor
- Less resilience
- No public access
- Pre-established (trusted) identities are needed. Bitcoin-type self-generated identities (addresses) are not possible