# BTA 2022 Homework Assignments

Version 1.0

Contact: Imre Kocsis (kocsis.imre@vik.bme.hu)

## General remarks

You have to choose a homework assignment from the set below and decide whether you want to solve the homework using Solidity or Hyperledger Fabric (using one of the supported chaincode languages).

**The assignments assume working in pairs. One person from the pair must register the team and select 3 preferred homework using the following Google Forms:**

https://forms.gle/8P8FaVtAbmxCudMP7

You may opt to work alone (subject to our acceptance), but please be aware of the fact that the cumulative expected "workload" remains largely the same.

All assignments are partially underspecified. You are expected to interpret the assignment and "fill in the blanks", and document these decisions (see Common requirements 1.a).

You are **NOT REQUIRED** to create a full "DApp" (a full distributed application that also has a client front-end implementation) – the homework focuses on the smart contract layer.

Please be mindful of the fact that most assignments require some form of participant management. The usual "style" of this differs on the different implementation platforms.

The same applies to handling time:

- In Solidity, you have only "block time": time passed is computed – roughly - from block depth difference and the roughly known block time of the consensus[1];
- In Fabric, you have a "transaction timestamp": the time when the client assembled the transaction proposal, thus this value will be the same on all endorsing peer nodes[2].

You are not required to deploy your solution "in production". The development environment recommendations describe the expected style of execution for testing.

Naturally, you can submit your homework in Hungarian – but we do accept English, too. (The reason that all our material is in English is obviously so that we don't have to translate everything every time when it's used in other contexts.)

[1] This is actually becoming a bit of an antipattern; for certain use cases, miners "playing" with the block depth they choose to mine a transaction at may enable certain attacks. That said, for the homework assignment, we fully accept block depth based time handling.

[2] And that can have its own problems, too, but in this context, we don't care about those :)

## Common requirements

You are expected to submit the following artifacts in a **single zip** file:

1. A **PDF** documentation detailing:
    a. the design decisions you took;
    b. the data model (structs/classes) that your contract stores and manipulates;
    c. the API of the smart contract;
    d. the important/non-trivial implementation details (if any);
    e. the definition and implementation of test cases;
    f. instructions for bootstrapping the project, running the test cases, and deploying the contracts.
2. A **directory** that contains your smart contract project (Truffle or VSCode project/workspace)
    a. The smart contract in a compilable and deployable form;
    b. The implemented test cases;
    c. **Without** generated artifacts or installed dependencies.

## Solidity specifics

For Solidity, we recommend submitting a Truffle project (see guide), which includes the contract(s) and the test cases and can be easily executed. The guide we created for the course:

https://ftsrg.mit.bme.hu/blockchain-ethereumlab/guide.html

## Hyperledger Fabric specifics

For Fabric chaincodes, we recommend using:

- the Visual Studio Code environment: https://code.visualstudio.com
- with an installed IBM Blockchain Platform extension: https://marketplace.visualstudio.com/items?itemName=IBMBlockchain.ibm-blockchain-platform

The extension also generates example unit tests using chaincode stub mocks, which can be easily adapted to your own chaincode's ledger access. **Unit testing** is the preferred way of testing. Transaction data-based "clickable" end-to-end "integration tests" are used to demonstrate more complex features/interactions. The extension also contains **embedded tutorials** for various topics on its home page (in VS Code).

If you have technical troubles with the environment, contact Attila Klenik (attila.klenik@vik.bme.hu).

## HF1 - Safe crossings for autonomous cars

Unguarded level crossings will pose a special challenge for autonomous vehicles; from the safety point of view, relying only on the on-board sensor packages and intelligence to decide whether it is safe to cross (no train is approaching) will be problematic. At the same time, the railway infrastructure – commonly partitioned into blocks (https://en.wikipedia.org/wiki/Railway_signalling#Block_signalling) – "knows" where the trains are; e.g., safety systems ensure that no train can enter a block already (or still) occupied by a train.

The task is to design and implement a train crossing smart contract with the following features:

1. The railroad infrastructure must periodically signal the crossing to be in a "FREE TO CROSS" state. This state has a preset validity time; if the last update is older than that, the crossing must be assumed to be in a "LOCKED" state. This can happen either on a train approaching or a failure of the infrastructure.
2. Autonomous vehicles wanting to cross must request permission to do so.
3. Permission may be granted only if the intersection is not in a "LOCKED" state.
4. Additionally, an intersection comprises one or more lanes. A single lane can accommodate a fixed number of crossing cars, predetermined by the railroad infrastructure managing authority.
5. Autonomous vehicles must explicitly release their permission after leaving the crossing. Failure to do so will later involve legal action; for this purpose, their identity must be recorded on the ledger, but in a privacy-preserving way (as much as possible).
6. As an additional safety measure, using a means of communication independent from the one used by the above-mentioned infrastructure, approaching trains also explicitly request the crossing to get into the "LOCKED" state (and release this signal only when they have passed it).
7. However, if the intersection is still occupied, it transitions into a special state (also signaling this to the train) where the train will have priority to request crossing, i.e., no more cars are granted crossing permission until the train crosses.
8. If the train can't gain permission in a predetermined time since the original trial, it is assumed that there is an obstacle in the crossing, and the train can break or halt.

Note: this exercise does reflect some of the concepts used in safety-critical engineering, but falls far from a full, real-life safety strategy, i.e., don't build a real system from this specification. If you happen to be a railway fan, the following references are good reads:

- https://arxiv.org/abs/1901.06236
- https://www.deutschebahn.com/en/Digitalization/technology/New-Technology/blockchain-3520362

**Homework owner: Attila Klenik** (attila.klenik@vik.bme.hu)

# HF2 - Course registration

Create a smart contract that can handle the creation and management of university courses by teachers, and allows students to register for the courses. The implementation must adhere to the following specification (the design of unspecified aspects is left to you):

1. The smart contract can differentiate between teachers and students, and performs the required access control of contract calls.
2. Teachers can create a new course with a given student limit and start date.
3. Students can register for the existing courses up to one week before the course start date.
4. If the course student limit is reached, then the student is placed on the waiting list of the course.
5. Teachers can increase the student limit for the course, updating the registration and waiting list appropriately.
6. Students can swap course registrations between each other (not sell, just swap existing registrations).
    a. Students can propose their registration for a swap (providing sufficient metadata).
    b. Other students can offer their own registrations for the proposal.
    c. The swap initiator can accept only one of the offers, concluding the atomic swap of registrations.
7. Teachers can close the registration period in the last week before starting the course (and not before).
8. Students and teachers can query the current state of existing courses freely.

**Homework owner: Attila Klenik** (attila.klenik@vik.bme.hu)

## HF3 - L(a)unch codes

A high security facility always houses a shift of two soldiers. It must also provide regular access to low security clearance staff (food delivery, cleaning, …). All entries and exits are tracked and authorized by a distributed ledger (a tamper-proof electronic lock on the main entrance continuously monitors the ledger and decides whether it should open or close; requests and authorizations are supported by smart cards and electronic terminals).

1. Entry is requested from the outside and must be authorized by both soldiers on duty.
2. Successful entry must be logged inside by the entering party (after the door is closed).
3. The protocol for exits is the same in the reverse.
4. Shift changes happen in two phases (first soldier1' replaces soldier1 in a full entry-exit cycle, then soldier2' replaces soldier2).
5. Guard duty is transferred inside by a mutual "acknowledgment" of the involved two soldiers.
6. There must not be more than 3 persons in the facility at any time.
7. Shift change must take place when the facility is empty, and no entry is allowed until it is over.
8. A soldier cannot enter or exit the facility while he/she is on guard duty.

Design and implement a smart contract supporting the above access management protocol.

**Homework owner: Attila Klenik** (attila.klenik@vik.bme.hu)

# HF4 - Private tic-tac-toe (*Advanced, Solidity only)

Most smart contracts do not have any privacy features, meaning every piece of data you upload will be visible to the public. One possible solution is to use zero-knowledge proofs.

Write a ZoKrates program (prover) along with a verifier/game manager solidity smart-contract, that lets you and your friend play tic-tac-toe, without revealing the current state of the game (e.g., the current marks on the board) to the public.

1. The game board must be at least 3 tiles tall and 3 tiles wide (3x3)
2. Each player must have a turn when they can place exactly 1 mark on an empty tile.
3. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row is the winner.
4. The current state of the game must not be stored on the blockchain (as plain text).
5. The prover must be able to prove that:
    a. A given move is valid (e.g., it is placed on an empty tile, it is their turn, etc.)
    b. It is a continuation of the previous state
6. A player can only place their mark if they can provide valid proof for the verifier.

Guide for ZoKrates: https://zokrates.github.io/gettingstarted.html

**Homework owner: Balázs Toldi** (balazs.toldi@edu.bme.hu)

# HF5 - Smart NFT: non-fungible weather token

Non-fungible tokens became quite popular these days. These tokens can represent a proof of ownership of an asset, but since they are written as a smart-contract, they can do much more!

Design and implement an ERC-721 compliant non-fungible token (NFT), that changes its image URL based on the current weather outside!

Key notes:

1. Each token shall be minted with a unique city (or district) name and a set of images for each weather condition.
2. The token shall have a function that gets the current weather of the city defined in the minting process.
3. Each time the weather data is updated the image URL should be updated accordingly.
4. Getting the weather data to the blockchain requires an oracle:
    a. In Solidity, you can use Chainlink (https://docs.chain.link/docs/request-and-receive-data)
    b. For Hyperledger Fabric, a different solution is needed, but it must not break the consensus mechanism. One possible solution is a participant that periodically uploads the weather data to the blockchain.

**Homework owner: Balázs Toldi** (balazs.toldi@edu.bme.hu)

# HF6 - Geocaching

Geocaching is a treasure-hunt-like public game where participants hide and attempt to locate hidden containers ('caches') around the world. Most players focus on finding more caches using their GPS coordinates and logging their visit – traditionally, with pen and paper, into a logbook found within the cache, as well as digitally on the geocaching.com website. Geocachers also place so-called 'trackables' into the containers, which can later be taken or swapped by other cachers. When a player inserts or removes a trackable into/from a cache, they include this transaction in their log post. This way, the location of trackables can be followed.

In addition to seeking caches, players are also free to become maintainers and hide their own cache. A cache (minimally) has a name, GPS coordinates, and usually some description, including a hint as to where the cache is hidden within the area.

The task is to write a smart contract to aid Geocaching. You are free to come up with your own blockchain-based Geocaching game, but take the following into consideration:

1. You must implement some mechanism that ensures only those who actually found the cache can log their success.
2. You must add support for trackables.
3. Only cache owners/maintainers may remove or modify caches. However, all players have the option to report problems with caches, e.g., if it seems to have gone missing or been damaged.

**Homework owner: Bertalan Zoltán Péter** (bpeter@edu.bme.hu)

# HF7 - Simple Blind Auction

Your task is to implement first-price sealed-bid auction (https://en.wikipedia.org/wiki/First-price_sealed-bid_auction) as a smart contract.

In this auction scheme, there are two phases: in the first phase, participants submit their bids **secretly**, so that nobody knows the bids of others (hence the 'blindness'). In the second phase, bidders reveal their bids, and the highest bidder wins and takes the item.

The item under auction is represented as an ERC-20 token. Requirements:

1. The seller must be able to specify the token under auction, the reserve price (ie minimum bid), and the durations of the bidding and revealing phase when starting an auction.
2. Participants can place their bids secretly by only submitting the cryptographic hash of their valuation of the item and a nonce. For simplicity, participants may only bid once. Bidding is only allowed in the bidding phase. Bidders must also attach sufficient funds to their transactions. Note that they should not be required to attach the exact amount, as this would effectively reveal their bids immediately.
3. Once the auction has entered the revelation phase, bidders are to submit their actual bid to the contract (an amount and a nonce), which must obviously form the pre-image of the previously submitted hash value.
4. After the revelation period, the highest bidder can claim the token. Bidders should also be able to withdraw any excess funds paid during bidding.

Do not forget to document how your solution handles problematic cases such as insufficient funds attached to bids and participants simply not revealing their bids together with your other design choices!

*For a more challenging task, you can try your luck with an incentive-compatible auction protocol, such as the Vickrey auction* (https://en.wikipedia.org/wiki/Vickrey_auction).

**Homework owner: Bertalan Zoltán Péter** (bpeter@edu.bme.hu)

# HF8 - Food chain

Traditional food supply chains often suffer huge losses when it turns out that the origin of the product is contaminated, because they must annihilate tons of potentially affected products, not to mention the costs of tracing them.

Your task is to design and implement a smart contract that allows tracking the route of each product from farms through factories and wholesalers to retailers. The system shall consist of the following parties:

1. farmers who register the resources on the blockchain,
2. shippers who register product transitions (e.g., from farm to factory, etc.),
3. factories that register products linked to the resources they are made from,
4. wholesalers who can split the chunks of products arriving from a factory into smaller units,
5. and retailers who can sell the final products.

If any parties mark one of their products spoiled, the system must trace and mark all other assets that may have been affected (typically the ones that originate from the same source, e.g., the ones that were shipped together). Parties can only mark products that are either produced by them or are currently in their possession. The contamination/defect report may also provide information about the type (location) of the defect: farm-related, factory-related, shipment-related, etc. If such information is available, the system should only trace the routes starting from the source of the defect (e.g., if a factory is the source of a defect, not all products shipped together from the same farm should be marked).

Note: This assignment requires the on-chain computation of transitive closure. Be aware of its implications (e.g., related to gas usage)! Naturally, we would most likely not do this directly on the Ethereum main net (where computation costs “money”), but the applicable techniques (one party computing the forward traces off-chain and providing a succinct proof of correctness, reaching agreement on the affected products, …) are beyond the scope of a homework. For consortial networks, this should be a lesser concern, but do provide an estimate (and possibly tests) for the problem sizes that are deemed manageable.

**Homework owner: Imre Kocsis** (kocsis.imre@vik.bme.hu)

# HF9 - Vaccination slot

Design and implement a “vaccination slot” utility token for a (single) medical station.

Vaccination slots are issued for specific occasions (at specific dates) to specific patients by the doctors. Patients are allowed to swap their slots with others, but only those patients can take part in these transactions who already own a valid token. Doctors “burn” the tokens as the vaccination of the owner occurs. No patient can hold more than one vaccination slot for any occasion at any time. Provide facilities for the patients to be able to “swap” vaccination slots in an atomic way. Take into account that the doctors use multiple types of vaccines, some of which have to be administered multiple times, within some (vaccine-dependent) time interval after the first shot.

Note: this token is kind of a non-fungible one; take into consideration that what part of the full ERC-721 specification is necessary for it. For Solidity: a full ERC-721 compliant solution is looked on favorably, but it’s not an absolute necessity. For Fabric, note that that you can easily “port” the ERC-721 interface specification.

**Homework owner: Imre Kocsis** (kocsis.imre@vik.bme.hu)