# Compiling and Deploying the Contract:

* Copy the solidity Smart Contract code given in the file ShipmentTrakcer.sol. This is in the Master Node directory “ExactPath\MasterNode\contracts”.
* Use the online Remix IDE to compile the contract. Please set the compiler version for Remix to version:0.4.25+commit.59dbf8f1. I have done my testing on this version as it was the latest version available at that time. Also some keywords are deprecated in later versions of the compiler so the code might not compile in the latest version of the compiler.
* Please turn on compiler optimization on Remix IDE otherwise it will give errors like file size too big during contract deployment.
* Once compiled we need to copy the ABI generated for ShipmentTracker contract. This abi needs to be placed in the file ABI.json in the directory “ExactPath \MasterNode\ShipmentTracker-DAPP\public\ABI”. I have already placed the correct ABI in the zip file that I will email you.
* Please deploy the contract on Ropsten Test network using the Remix IDE. The Ethereum key that deploys the contract becomes the Master Key for the Master node for later use. Please don’t change this key when running the Master node.
* The contract is deployed using two transactions. One transactions deploys the library contract and the second transaction deploys the actual ShipmentTracker contract. Metamask will prompt you to sign the transactions please sign both.
* Once the contract is deployed you will get a contract address. Please copy it for later use in the Master node and IoT Node.
* Sometimes you might see gas estimation error when deploying contract on Ropsten. This is a weird bug with Remix, It can be easily resolved by changing the current network in metamask to rinkeby and then back to Ropsten. After this try deploying the contract again it should work now.

# Pre Requisites for Master Node:

**Master Node:**

* We need to install Node JS to run the code for the master Thesis. I did all my testing on Windows 10 machine but the code should work for any machine that can run Node JS. However, some minor directory changes might need to be made in order to get the code to work on linux machines.
* Please install Metamask browser extension for communicating with the Ethereum Network on the browser which will be used for running the Master Node App. I used Google chrome for all of my testing but it should work on firefox as well.
* In order to send signed transactions, you will need to generate Ethereum keys. Metamask allows you to generate Ethereum keys as well.
* The testing was done on Ropsten test network, so we need some test ether to pay for signed transactions. There are several faucets out there that will allow you to request test ethers for ropsten. Some that I have used in the past are <https://faucet.metamask.io/>, <https://faucet.ropsten.be/>

# Running the Master Node:

* Please use a terminal or cmd to change to the base directory for master node which contains the app.js file. It should be something like this “ExactPath\MasterNode\ShipmentTracker-DAPP”.
* Please install all dependencies by running **npm install**. (**Note:** Ideally this should work without a problem but I do remember the very first time when I installed the dependencies on my pc I ran into problem as node-gym etc was giving some problems during installation. I googled the errors that were coming to resolve them easily by installing some more dependencies. I tried to reproduce this issue again but it does not appear for me anymore.)
* Please run the MasterNode using **npm run**.
* Using a browser preferably google chrome goto “http://localhost:3000”.
* You will be presented with a login screen. In the contract address field please paste the address of your deployed contract and press login.
* After logging in you can decide on shipper rights, and also define shipping requirements

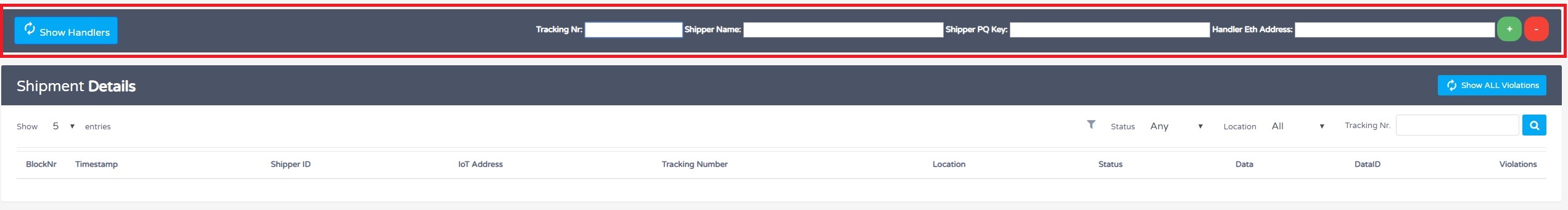
## Sub Module Main

## Defining shipper rights:

Please see the attached screen shot, after filling the fields in the red box click the green ‘+’ button to enable an IoT Node to send violation to the SmartContract.

There are four fields to fill: TrackingNR, Shipper Name, Shipper PQ key and Handler Eth Address

As we discussed master node can define any unique values for these fields and the shipper / supplier must configure the IoT node according to the fields defined by the master node.



**Tracking NR:** A string that uniquely identifies the current shipping cycle. Use any unique string for this. I have hard coded one tracking number in the code (main.js) for the IoT node. If you define a new tracking number also change it in the IoT Node. I used this tracking number

**Shipper Name:** This is the shipper ID and should be unique for each shipper within a tracking cycle. I have hard coded two shipper IDs in the code for IoT node for testing. if you define new ids using the master node please also change them in the IoT Node (main.js).

**Handler Eth Address:** This is the public Ethereum key / address of the IoT node that sends signed transactions to the smart contract.

**Shipper PQ key:** Use any 256 bit key for the simulated PQ algorithm. But this key should be the same as in the IoT node. I have hard coded two PQ keys for the two shippers in the code of the IoT node for testing. If you use a different keys here please also change those keys in the IoT Node (main.js).

**Sample values for these fields:**

**Sample 1:**

07b323db

DHLDarmstadt1

eff3bc952afdc46400bcfc07a5699f525119760f364cb04129323e207fcdc18c

0x58deecc0b671b8fc83b81b6153ad3565de430f03

**Sample 2:**

07b323db

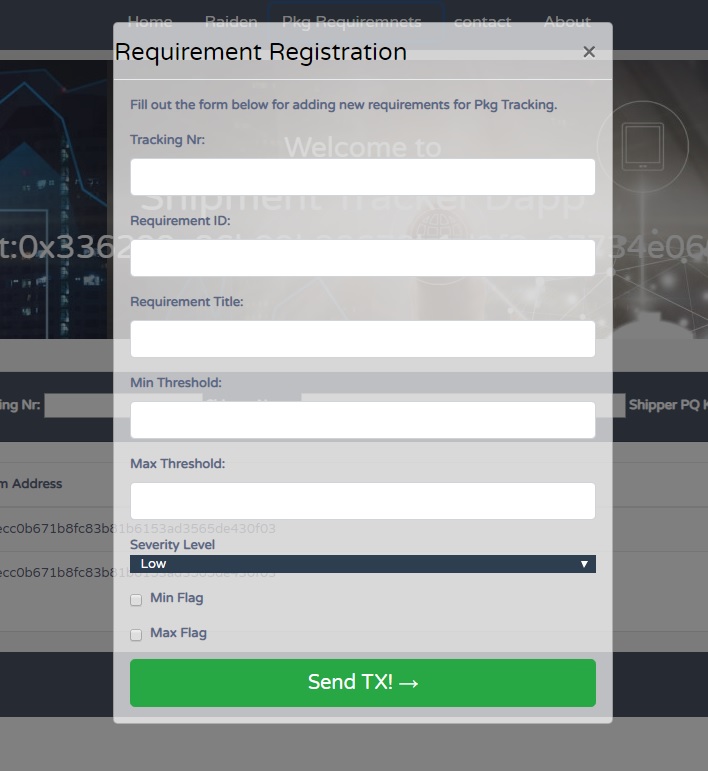
DHLDarmstadt2

d6cea69138fa00da0fa2af65912191ad45b35a690150855e99c2f3c293334f85

0x58deecc0b671b8fc83b81b6153ad3565de430f03

## Defining Requirements

Please see the attached screen shot below:



* You can click the package requirements tab to define requirements for individual Tracking numbers / IoT Nodes. Please note that the fileds Min and Max threshold expect unsigned Int values.
* The Requirement ID is used by the IoT node to identify, monitor and configure monitoring logic for individual requirements. As discussed we have upto 4 unique requirement IDs i.e.
  + “TEMP” for Temperature sensor
  + “HUMIDITY” for the Humidity Sensor
  + “LUX” for the light sensor
  + “dB" for the loudness / sound sensor. (set this to a higher value to avoid triggering repeated violations because of ambient noise)
  + Optionally “PRESSURE”, this takes dummy hard corded values for testing purposes. Our hardware development kit did not come with a pressure sensor.
* The flags are used to indicate to the IoT node which threshold is relevant for a particular violation e.g if “TEMP” requirement was defined with Min and Max threshold between 10 and 35 degrees and both min and max flags were checked. In this case the IoT node will send a violation to the smart contract if package temp falls below 10 or goes above 35. If only max flag was checked only max threshold value will be checked for the requirement ID.

## Show All Requirements:

The show all violations button can be used to see all the validated violations which have been communicated and stored in the blockchain. Optionally you can use the browser console to see in real time each time a violation event is fired from the smart contract.

## Sub Module Raiden API

As a reminder please note that Raiden client and local geth client must be up and running before using this module. Geth should be fully synced with the Ropsten Test network. I tested this module separately due to all the problems faced getting Raiden up and running. I used the raiden version 0.1.0 and 0.0.9 for my testing.

As discussed this the GUI module which gives us access to the Raiden API. Raiden comes with its own GUI as well but I integrated a gui module for making API calls to raiden API into our master node. As communicated earlier Raiden is not fully stable and doesn’t always work because its NAT punching module does not always goes through firewalls and NAT routers.

Raiden uses ERC20 Tokens. I have placed three ERC20 token contracts in the folder at “\MasterNode\contracts\ERC20”. I think TOKEN.sol is the latest one. Deploy the ERC20 token use its address for Raiden client and establishing channels.

Please see the instructions on this link to get raiden client up and running:

<https://raiden-network.readthedocs.io/en/stable/overview_and_guide.html#installation>

Raiden API calls are explained here:

<https://raiden-network.readthedocs.io/en/stable/rest_api.html>

My own Notes for configuring and using raiden are given in the file VirtualEnvironment&more.txt and RaidenAPITEST files. Some important things are described below.

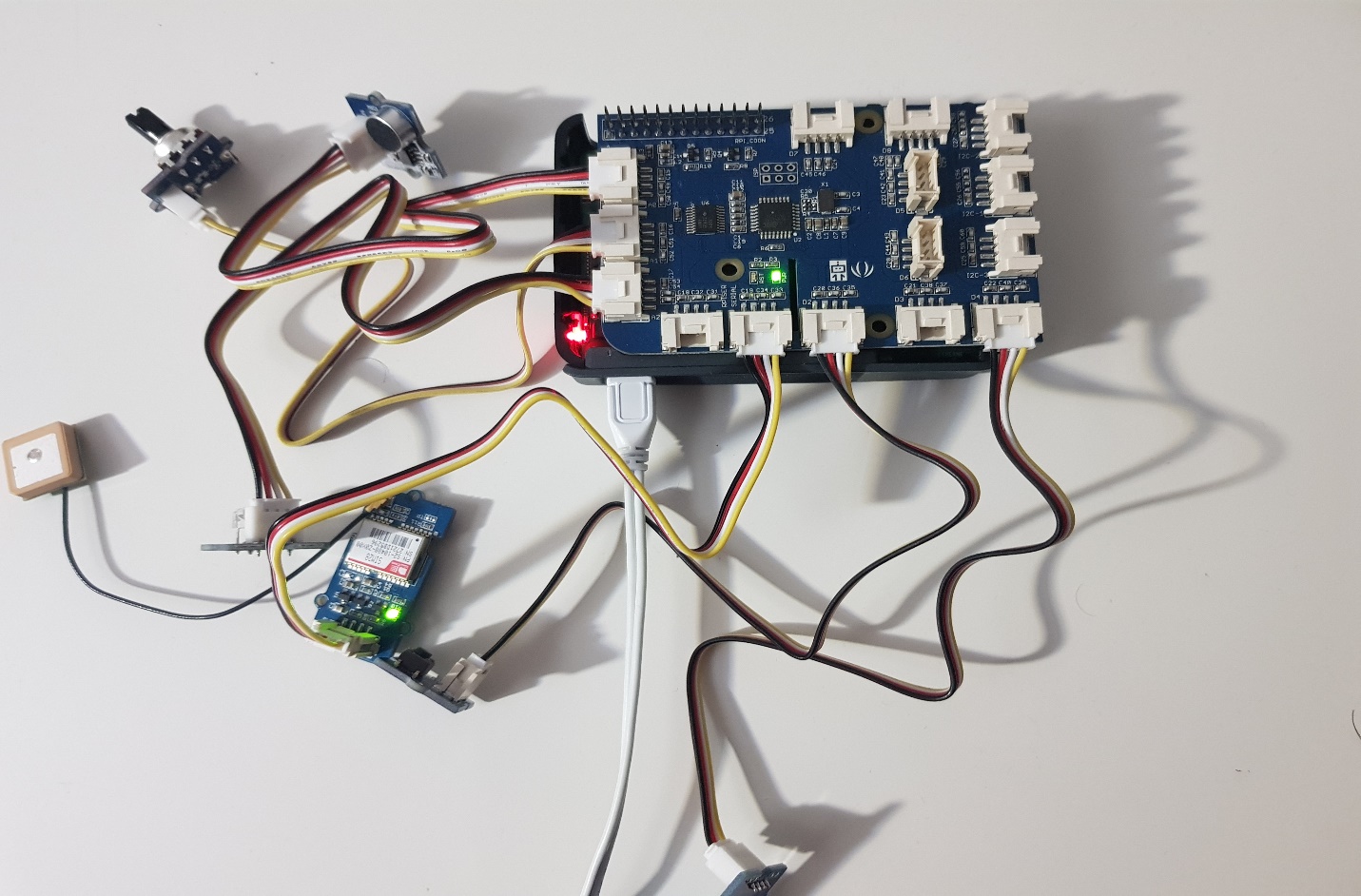
* As I explained during my raiden demo, the two nodes need to see / ping each other in order for direct transfers to work.
* I found that turning off all firewalls including windows, and any router /modem firewalls helps for the pings to go through.
* If the two nodes are not able to ping each other try restrating the raiden clien
* If restart doesn’t work, try **raiden removedb** and restart raiden client again.

# IoT Node:

**Hardware Setup:**

* Please attach the Grove PI Board to the raspberry PI
* Attach loudness sensor to port A0
* Attach rotary angle sensor to port A1
* Connect the Light Sensor to port A2
* Connect the Push button to Port D2
* Connect Temp and Humidity sensor to port D4
* The GPS sensor connects to the serial port.

A correctly configured raspberry pi is shown in the following figure:



**Software Setup:**

I tested the IoT node by running it on “Raspbian GNU/Linux 9 (stretch)" and node version v9.11.2. My npm version on the raspberry pi was 5.6.0

**Useful links:**

<https://www.raspberrypi.org/documentation/installation/sdxc_formatting.md>

<https://www.raspberrypi.org/forums/viewtopic.php?t=178593>

<https://www.instructables.com/id/Install-Nodejs-and-Npm-on-Raspberry-Pi/>

<https://thisdavej.com/beginners-guide-to-installing-node-js-on-a-raspberry-pi/>

**Pre Requisites:**

* RaspberryPIs must be running Raspbian OS. The raspberry PIs I will give back are already configured with the proper OS.
* Copy the directory titled “RaspberryPI” from the zip file in the raspberry PI folder.
* Install Node JS and Npm on the raspberry pi. Have a look at the useful links above if you are configuring it from Scratch
* Please paste the compiled ABI in the ABI.json file in the directory “\RaspberryPI\ShipmentTracker-DAPP\public\ABI”
* Please paste the contract address in the contract.txt file in the directory “\RaspberryPI\ShipmentTracker-DAPP\public\ABI”
* You need a valid ethereum key pair to send signed transactions to the blockchain, I generated extra keys on my windows machine using metamask and then copied the keystore file on the raspberry pi.
* Alternatively, you can create ethereum key pairs directly using geth but then you first have to install geth client on the raspberry pi.
* Please note that this key should have some test ethereum assigned to it for sending signed transactions to the ropsten network.
* Please sign up to infura for accessing ropsten and IPFS network from the raspberry PI. You will be emailed API access keys for infura clients.
* Paste your unique infura url for Ropsten in the Utils.js in the following function

web3 = new Web3(new Web3.providers.HttpProvider("https://ropsten.infura.io/APIToken"));

**Running IoT Node**

* Open a terminal on the raspberry PI and change the working directory to where you placed the code provided in the “RaspberryPI” directory.
* Change the working directory to “\RaspberryPI\ShipmentTracker-DAPP” i.e. the directory that contains the app.js file.
* Before running the node we need to setup few variables, please use a text editor to open the app.js file and provide your own values for the following:

**var pubAddress** = //public key address of the Eth key generated for this node. Remember to give access to this key from master node.

**const password** = //password to unlock your keystore file.

**const datadir** = // path to your keystore file the default path is usually "/home/pi/.ethereum/testnet";

* In the main.js file set the tracking number on line 25 i.e. IO.setTrackingNr("07b323db"); // change "07b323db" to any tracking number this IoT Node has access to. This access was configured earlier using the Master Node.
* Two shipper names have been hard coded you can change them if you want. Locations to change , main.js (line:30) and Sensors.js (Line:37 and 41)
* Run “npm install” //wait for all dependencies to be downloaded and installed, this can take some time.
* Run “npm start” to start the IoT node monitoring process.

**Push button**

* As I explained during the Demo / thesis defnese push button is used to simulate shipper change condition. A short press of push button changes the current shipper. Sometimes the push button causes the IoT node to crash. In this case make sure the push button is properly free and at rest position. I find pushing the push button when the IoT node software is not running resolves this issue.
* Long press of the push button simulates the package delivered condition and graceful close of the IoT node monitoring software.

**RotaryAngle Sensor**

* As described during the thesis defense / demo rotary sensor is used to simulate the location change. Rotating the Rotary sensor changes the current city.

# Final Remarks

Most of the above mentioned instructions are related to the case when everything is setup from scratch. However, In the case that Master Node and IoT Node are properly setup and configured a user only has to run the following two commands:

Npm install

Npm start