**INTRODUCTION TO MOTION2NX**

# 

# Table of Contents

# 

[**Table of Contents**](#_4tj8on9q687w) **0**

[**INTRODUCTION**](#_jhosz9fyrc6) **1**

[**THE ROOT FOLDER**](#_k0c7demkr7u1) **1**

[**THE SOURCE FOLDER**](#_fa78xs8ojxst) **2**

[**THE MOTIONCORE FOLDER**](#_rbll19r8aoco) **3**

[Algorithm](#_at729kdojr6) 3

[Base](#_5u2n1fojwtme) 3

[Cmake](#_2z92tjy8b7u8) 3

[Communication](#_vob5gy36i1i1) 3

[Crypto](#_cqshp1rb3hc2) 4

[Data\_Storage](#_525dyc9uxtpe) 4

[Executor](#_l8306zmnvrvf) 4

[Gate](#_jys5a9ivvo6) 4

[Protocols](#_k7r9xvzcix5n) 5

[Secure\_type](#_qgkr6fz72tu8) 5

[Share](#_u5f8acxx9qkm) 5

[Statistics](#_bjq5a57w5vmr) 5

[Tensor](#_mjje7xggjegx) 6

[Utility](#_d8eczhp660zb) 6

[Wire](#_wbm2by7i3q1j) 6

# 

# 

# 

# **INTRODUCTION**

The MOTION2NX repository was created for the primary purpose of executing two party protocols such as ABY2.0, GMW and so on. The speciality of the repository lies in the fact that a majority of the operations compliant with the two party setup can easily be executed due to the users having the freedom to choose their own protocols, run on their own systems as well as set their own types of inputs. Furthermore, it creates a sense of security as well as tracks any errors and mistakes, making it easy to backtrack on any issues caused using the log files. Along with operations and implementations, the repository consists of numerous examples pre-created as well as freshly added that exploit a majority of the operations present in the repository. The following documentation does not explain the list of operations, but rather guides the reader on where to find what types of operations and how to execute certain examples.

# **THE ROOT FOLDER**

The root folder consists of the README used to install the repository. A more detailed explanation for the same can be found in the documentation for Yao’s millionaire’s problem. The list of files present are:

1. **Circuits:** This contains a list of **bristol** files, and these are used to build multiple gates put together in an orderly fashion to form a circuit. Different circuits exist, such as those to perform a simple greater than operation, to perform basic arithmetic operations ( +, -, \*, / ) and so on. Further, they are created based on the input sizes, ranging from 8 bit values to 64 bit values.
2. **Cmake:** Contains the initial files required to create and build the repository from scratch.
3. **Docs:** Documentation on the upgrade from an ABYN setup to MOTION
4. **Extern:** External support packages from googletests, benchmarks, etc.
5. **Fbs:** Flatbuffer files used by googletests for all protocols
6. **Src:** The heart of the entire repository, contains the examples as well as the implementations of functions and operations.
7. **CmakeLists.txt:** Requirements for creating the build folder to execute examples and activate operations.

# **THE SOURCE FOLDER**

As mentioned earlier, **src** is the true source file for the entire framework, and consists of four files:

1. **Contrib:** Contains the extra contributions for the HYCC and ONNX adapters
2. **Tests:** Holds various test executables, tests for functionalities present within the repository.
3. **Examples:** List of all example files that can be executed upon building
4. **Motioncore:** The core folder that contains all the files necessary for operations.

The steps to setup the repository have been mentioned in the Installation Guide. Once we have the setup, we simply need to head to the build\_ folder to execute the examples, one of which is the Yao’s millionaires problem. Other examples are executed in a similar fashion, where on a local system, two terminals are booted up and each one is treated as a party. For each, we add the following common flags:

1. **--my-id:** Indicates the ID of the party in use
2. **--party:** Used to enter the partyID, IP and Port number of all parties participating in the process
3. **--arithmetic-protocol:** Choice of arithmetic protocol
4. **--boolean-protocol:** Choice of boolean protocol
5. **--repetitions:** Number of times the program repeats execution
6. **--num-simd:** Number of Single Instruction Multiple Data values.

Apart from the ones listed above, more flags exist specific to each type of example. Since the concentration is primarily on the millionaires problem, the documentation for the same covers the execution of the example as well as the meaning behind the entire code. We shall dive into the **motioncore** folder to explore the contents of each file and what they are used for.

# **THE MOTIONCORE FOLDER**

The motioncore folder consists of numerous sub-folders, each holding its own importance. Each one contains certain codebases, functions, structures and supporting programs that aid in the execution of all the examples. Here we shall get a top level understanding of the contents within each of these sub-folders.

## **Algorithm**

Not the most commonly used of the folders, this one focuses on providing a support system during the circuit creation process. The list of files present here give the definition of certain algorithms and steps that the underlying programs must use in order to build a fully functioning circuit given the choice of operation, protocol and inputs.

## **Base**

If the Algorithm folder provides instructions on how to build circuits from the Bristol files, the base folder provides all the tools necessary to arrange and construct each level and gate of a given circuit based on the algorithm used. Further, this folder also provides a platform to initialise an array of operations by constructing a temporary backend for a Two Party Setup. These programs take in all the inputs given by the parties in the examples and set them up along with the necessary function calls to process them based on the instructions provided. Overall, the base folder creates the first building block prior to beginning the execution of any of the examples in the repository.

## **Cmake**

A minor folder, it contains a single file that specifies the version of the repository, the versions required for the repository to run smoothly and the details of other dependencies, similar to YAML or any kind of setup file.

## **Communication**

One of the most important processes prior to performing operations on inputs is to allow the two parties to connect and communicate with each other, which is precisely what this folder provides us with. Files such as the base\_ot\_layer, ot\_extension and sync\_handlers are responsible for providing support using oblivious transfers for the parties to exchange vital information in a secure manner. The files communication\_layer and tcp\_transport are responsible for providing an asynchronous tcp connection between the two parties. Other files such as hello\_message and message handlers are responsible for overlooking the encapsulation and security of the data being transferred between the parties.

## **Crypto**

An interesting folder, this is responsible for providing a vast array of encryption schemes to be used based on the functions that call for them. AES keys, Oblivious Transfer encryptions, Random value generators and many more functions exist as a part of this folder. Three important files, pseudo\_random\_generator, sharing\_randomness\_generator and the motion\_base\_provider are responsible for the generation of shares in the case of ABY2.0 and GMW examples.

## **Data\_Storage**

< requires completion > Basic folder used to store and test based on the protocols and providers used for executing certain examples.

## **Executor**

< requires completion > Used to create variants of original function execution formats for more efficient and faster processing.

## **Gate**

The gate is one of the most basic building blocks of any operation performed in the repository. From accepting inputs to multiplying numbers to sending and receiving data within parties, gates are abundantly used, and each one is created with n-number of wires for inputs and outputs and the operation that is to be performed by it. The gate.cpp file serves as the heart of all these operations, and override files for gates are present within every individual protocol that execute their gates in different ways unique to their method.

## **Protocols**

The deciding factor in every example, different protocols provide a different array of operations based on the regulations set by each one of them. The most commonly used one is **beavy**, another name for the **ABY2.0** protocol, and this extensively makes use of all the other folders mentioned, along with their variations such as in the case of gate.cpp or tensor folders to make them protocol specific.

Each protocol has its own unique way of handling the input values. Beavy makes use of complex delta shares in all of its operations while GMW uses simpler additive shares that do not require extra steps per operation to execute. Yao protocols on the other hand require **garbled circuits**, completely differing from the shares operations of the other two. For all these protocols, functions specific to the execution and handling of data have been created and placed in their respective folders.

## **Secure\_type**

< requires completion > A minor file detailing the instructions for building a special type of circuit.

## **Share**

This consists of multiple programs that test the creation of different types of shares based on the protocols used, the type of inputs being entered as well as the num\_simd value mentioned in the flags for each example. The generation of normal shares takes place within the folders for the respective protocols mentioned above.

## **Statistics**

This folder is responsible for calculating the runtime statistic values for each example, ranging from the amount of computing resources used, the amount of memory taken up by each step of the program, the time required for the execution of operations and so on. The statistics are compliant with the logs being recorded simultaneously, and are displayed in tabular form on the terminal.

## **Tensor**

Tensors are mathematical storage units, and for the repository, an entirely new implementation specific to the protocols has been created in this folder. More details for the same can be found in the Tensor documentation provided.

## **Utility**

One of the easily ignored folders for its importance, this crucial folder is responsible for defining the building blocks for each type of operation, and adheres to minor details by handling exceptions and errors as well. To give an example, to perform a multiplication between two tensors using beavy, we need to create the shares, where files like bit\_vector aid in the random key generation process, fiber\_condition monitors the operation timings for each wire ( all operations are in the form of wires ), logger files that store the details of every step of the runtime, helpers that provide special key values for encryption and so on.

## **Wire**

Holding the functions used to define how the data is stored, the wire folder contains a variety of functions using which values such as inputs, intermediates and outputs are stored and processed. Each wire is a special type of structure holding not only the values, but also details related to them.