Research Report Fall 2023

Kai Adams

Graduate Mentor: Shane Kosieradzki

Primary Investigator: Dr. Ueda

**Introduction:**

The goal of this project is to simulate encrypted dynamic systems. This research could be applied to many industries, including automotive, aerospace, and defense, which all may aim to utilize digital twins to reflect, model, and predict future behavior. The automotive industry would aim to implement these into cars in order to have functioning self-driving features. These could be susceptible to cyber attacks in the form of spoofing, sensor tampering, remote hacking, etc. [1]. The aerospace industry would use digital twins as system models in flight controls. This presents vulnerability to the flight controls in the forms of cyber attacks which could potentially change flight paths by taking control of the flight simulations or feeding misinformation to the system. Encryption would be paramount to the future safety of these industries. An encrypted model would aim to keep the information of the system model safe.

To model these systems, we can use a Functional Mockup Interface (FMI). FMI is an open standard for the organization of data for a system its governing differential equations. The FMI has three different configurations, but two are useful for industry applications – Model Exchange (ME) and Co-Simulation (CS). CS configuration, when exported, includes a solver, provided by the manufacturer of the software which the model was exported from. The Model Exchange configuration allows the implementation of the users’ own solver. This is a key feature of exporting and using Functional Mockup Units in an encrypted setting. By programmatically implementing an encrypted solver, encrypted system models could theoretically be operated on while remaining safely encrypted. The aim of this project is to create and implement an encrypted solver in such a way that encrypted FMU could be passed through and operated on.

A diagram of a function

Description automatically generatedA diagram of a function

Description automatically generated

**Figure 1**: Model Exchange (left) vs Co-Simulation (right) as displayed in the FMI standard.

Modeling dynamic systems requires consideration of every part of the system. While accurately modeling these systems from scratch is possible, there are many tools which are capable of creating the models, so we aim to leverage existing tools to streamline the process. We may use high level tools, such as Simulink, or some FEA software to do this. However, the problem with using these tools is that they are proprietary, so we do not have access to the source code, as it is considered intellectual property. Because of this, we are unable to incorporate our encrypted solver with the tools as they are. Instead, we can export models created by our high-level tools as FMU and utilize the open standard called FMI in order to use the aforementioned encrypted solver on our simulation. We are able to use the publicly open FMI standard to write software in order to accomplish this.

**Progress:**

The first system modeled was a damped harmonic oscillator, which was modeled as a mass-spring-damper system in Simulink and exported as a CS FMU.

A diagram of a mathematical equation

Description automatically generated

**Figure 2**: Mass-Spring-Damper system model.

Using Simulink, the values for the mass, spring, and damper constants could all be altered. The output of the system which was examined was the position of the mass. This system was then altered to have an input force and to make the scope an output readable within the FMU.

A diagram of a complex function

Description automatically generated

**Figure 3**: Mass-Spring-Damper system with readable input/output.

Utilizing FMPy, a python library used as an interface for running and editing FMU, the parameters of the system were programmatically altered, and the input force was made to be a step function. Which had a value of 0 at t < 1s, and a value of 1 after.

A screenshot of a computer program

Description automatically generated

**Figure 4**: setting the physical parameters and input through python script.

A screen shot of a graph

Description automatically generated

**Figure 5**: Graphical results from physically altered parameters.

The next step was to create a multiple input multiple output model, so a double damped harmonic oscillator was created in the form of a two mass-spring-damper system in Simulink. The velocity and position of both masses are the measured outputs of the system, and there is an input force on one of the masses. This model was exported as a FMU, and was simulated using the python script. Similar to the single MSD system, the two mass-spring-damper system was programmatically altered and graphed to see the change over time. Initially, the input force was the only changed value, and was made to be a step function. It was then set to a later time, so the system would settle from its initial positional excitation before being exposed to the input force. After, the masses were changed to also step with time, and the input force was set to be similar to an impulse force.

A diagram of a mathematical equation

Description automatically generated

**Figure 6a**: Drawn representation of 2MSD system with input force.

A screenshot of a diagram

Description automatically generated

**Figure 6b**: Simulink model representing the 2MSD system.

A graph with lines and dots

Description automatically generated

**Figure 6c**: Graphs of 2MSD system generated by FMPy.

A black screen with many small white and blue text

Description automatically generated with medium confidence

**Figure 6d**: Variables within the 2MSD FMU as shown in the model description xml.

A screen shot of a computer code

Description automatically generated

**Figure 6e**: Python script altering the physical parameters of the 2MSD system.

A graph with lines and dots

Description automatically generated

**Figure 6f**: Graphical results after changing the input force programmatically.

A screen shot of a computer code

Description automatically generated**Figure 6g**: Altering the mass to change dramatically at t = 75s.

A graph of a graph

Description automatically generated with medium confidence

**Figure 6h**: graphical result with significant mass change at t = 75s.

Within the CS solver, the time steps are strict. Since the solver is a part of the FMU when it is exported in the Co-Simulation format, the time steps must be rather large and, in the case of the model shown, divisible by 6. This results in a rougher-than-desired plot of the system simulation results.

After programmatically altering multiple CS FMU using a python script, the project moved towards the usage of ME FMU. However, Simulink, the tool which had been utilized to create all of the models so far, is unable to export FMU in a Model Exchange configuration and can only export Co-Simulation FMU. Because of this, Scilab was used to generate a new model. Scilab’s ‘Xcos’ is similar to MATLAB’s ‘Simulink’, as it can create models of dynamic systems. Using a toolbox known as the ‘fmu\_wrapper’, xcos can be used to export FMU. Scilab also has the benefit of being open source and free, though the user interface leaves much to be desired. First, scilab was used to generate a mass-spring-damper model similar to the model created in Simulink.

A diagram of a block diagram

Description automatically generated

**Figure 7**: Xcos model of a mass-spring-damper system.

The fmu\_wrapper documentation states it is also able to read FMU programmatically and read the data within. However, when run, the external FMU was read and the model description was generated, but other FMI functions were unable to run without crashing, regardless of the approach.

A screenshot of a computer program

Description automatically generated

**Figure 8**: model created in Simulink being read within Scilab.

**Future Work**:

In order to generate models which can be used with an encrypted solver, we aim to find a high-level tool which is able to export ME FMU seamlessly. Additionally, we aim to create an encrypted solver, which would be able to take in an encrypted FMU and alter it programmatically without decrypting the system model. We would aim to use this to model and predict physical systems securely. We aim to implement an encrypted system and solver into a microcontroller to securely simulate, predict, and control a physical system.

References:

[1] <https://www.cyres-consulting.com/autonomous-vehicle-cyber-security-overview/#:~:text=Due%20to%20their%20heavy%20reliance,by%20exploiting%20a%20software%20vulnerability>

[2] FMI Specifications 2.0.4