# Proof of Concept of MMT SCSCP Server

Victor Vasilyev

August 8, 2017

## 1 Concept

The idea behind an MitM SCSCP server is abstraction of abstract algebra systems from the users in order to make the use of those systems more accessible. In order to prove the possibility of this concept, we are developing a system consisting of the following parts:

- Central MMT SCSCP server that acts as an interface between a user and advanced algebra systems
- GAP SCSCP server that specializes on group theory
- Singular SCSCP server that specializes on polynomial calculations
- Python client that uses the SCSCP package

*Figure 1: Structure of the Proof of Concept System*

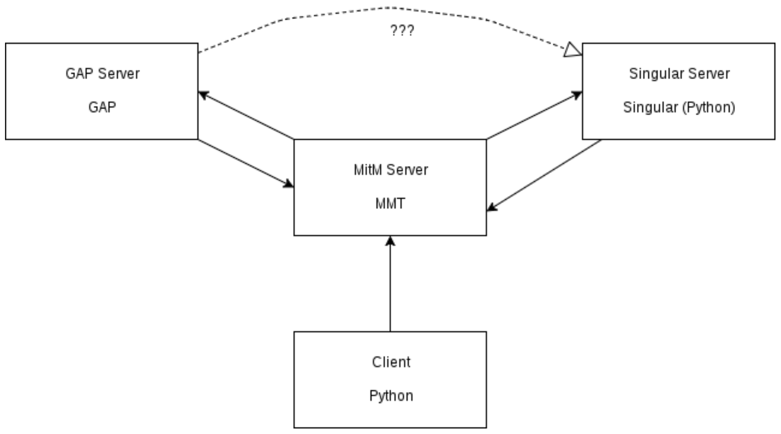

Figure 1 illustrates the structure of the system. The dotted arrow indicates potential interaction between GAP and Singular that bypasses the MMT server for efficiency.

## 2 Development

### 2.1 Formalization of Group Theory Concepts

In order to ensure smooth interaction between different algebra systems, the MMT server must have access to formalized concepts represented by them. The first step was thus to formalize the concepts of group theory available in the GAP system in MMT. The resulting code can be seen in MitM/groups repository on Mathhub[1].

### 2.2 Establishing a GAP Server and a Python Client

To familiarize myself with the Python client API, I ran a GAP server with minimal functionality[2] and queried it using the Python client[3].

### 2.3 Establishing Connections to MMT SCSCP Server From Python and GAP

For the architecture described in Figure 1 to work, the MMT server has to be able to provide service to both GAP and Python SCSCP clients. To verify this ability, I have used the server example provided with the MMT distribution[4] that exposes a single symbol- addition- and queried that using both GAP and Python. After applying a small patch to the MMT SCSCP server, that worked from both sides.

### 2.4 MitM/SCSCP Protocol v1.1

In order for the architecture in subject of this project to be extendable, the SCSCP servers have to be fully modular, and the MMT server has to accept new algebra systems and present their functionality to the end user during runtime. SCSCP inherently does not provide such functionality, so I propose a protocol implemented over SCSCP, namely MitM/SCSCP (see below).

### 2.5 MitM/SCSCP Protocol v1.2

Although the initial protocol provided functionality for adding multiple CAS to the MitM server, there was no alignment between them, i.e. if two different CAS exposed a symbol called "set", there would be no relationship between them. MitM thus acted as no more than a router for SCSCP requests and was not very useful. In order to create a fully abstract symbolic computation interface, I have redefined the protocol to v1.2. According to this protocol, symbols exposed by CAS have to be aligned with symbols from global CDs. This enables different CAS to reach an understanding between symbols they share.

# 3 Mitm/SCSCP Protocol v1.1 Specification

## 3.1 Peers

Although MitM/SCSCP attempts to establish a peer-to-peer connection over the strictly client-server SCSCP[5], one of the systems in the dialog must initially act as a server. This server will be called the MitM server and expose a strict set of function headers to all clients. The other system will be referred to from now on as CAS (Computer Algebra System) and can expose an arbitrary set of function headers.

## 3.2 Interaction

The MitM Server must, upon boot-up, expose exactly two functions over SC-SCP: *registerServer* and *removeServer*. Both functions must take exactly one argument: the address at which the CAS server being registered can be reached. Upon received a process instruction to *registerServer*, the MitM server must then perform a handshake with it and request a list of headers. After the handshake, the MitM should expose the headers of the CAS as part of it's own handshakes with external clients. When the CAS is going offline, it should call *removeServer* with it's address to safely shutdown the connection, at which point the MitM server should remove the headers of that CAS from the list of headers it exposes.

*Figure 2: Illustration of the MitM/SCSCP Protocol*

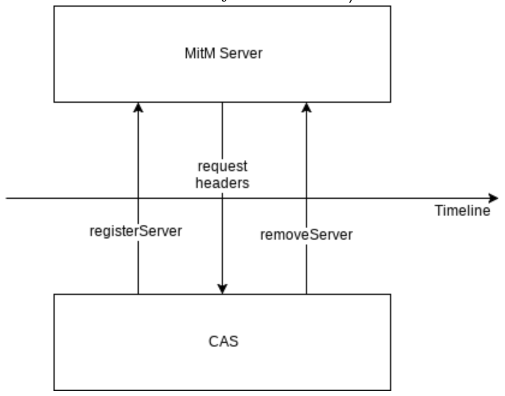

Figure 2 illustrates the skeleton conversation in MitM/SCSCP. Between requesting headers of the CAS and receiving a request to shut down the CAS, the MitM server can receive requests for the headers given by the CAS and pass them on to it.

## 4 MitM/SCSCP Protocol v1.2 Specification

### 4.1 Peers

The communicating sides remain similar to v1.1. In this specification, however, I shall also define two types of clients: a controlling client, that sets up the communicaion between servers and the alignment of symbols, and the naive client that is unaware of the CAS servers and simply performs symbolic computation requests to the MitM servers.

### 4.2 Interaction

As well as the functions exposed by the MitM server in version 1.1, in v1.2 there are three additional functions: *getAllServers*, *registerFunction* and *removeFunction*. They do the following:

- **getAllServers** takes no arguments and retrieves a list of names that correlate to the CAS instances that the MitM server is aware of.
- **registerFunction** takes three arguments: the name of a CAS SCSCP server, the symbol exposed by that server, and the global symbol that the CAS function is being aligned with. It then adds the global symbol to the list of function headers it exposes to clients and, when a client makes a request for that symbol, calls the aligned function on the CAS server.
- **removeFunction** takes two arguments: the name of a CAS SCSCP server and a symbol it exposes that has been registered on the MitM server. After this function is called, the MitM server cannot make calls to that symbol on that CAS.

In the new version of the protocol, the MitM server, upon receiving a request to register a CAS server should NOT retrieve its headers. Instead, the headers have to be aligned with global symbols through calls to *registerFunction*.

## 5 References

1. MitM/groups repository
2. Example GAP SCSCP server
3. Python SCSCP package
4. MMT repository
5. SCSCP Specification