# Testing SCAPI

Contents

[Testing SCAPI 1](#_Toc282588668)

[Purpose: 1](#_Toc282588669)

[Package: 1](#_Toc282588670)

[Output: 2](#_Toc282588671)

[Timing: 2](#_Toc282588672)

[1. Communication layer. 2](#_Toc282588673)

[Checking results: 2](#_Toc282588674)

[Proper and simple behavior: 2](#_Toc282588675)

[Performance: 4](#_Toc282588676)

[Wrong behavior: 2](#_Toc282588677)

[Input size: 3](#_Toc282588678)

[Number of participants: 3](#_Toc282588679)

[2. Primitive layer. 5](#_Toc282588682)

[Sources: 5](#_Toc282588683)

[Wrong input: 5](#_Toc282588684)

[Platforms: 5](#_Toc282588685)

[Performance 5](#_Toc282588686)

[General design: 5](#_Toc282588687)

[Appendix 6](#_Toc282588688)

[Table of algorithms/ sources: 6](#_Toc282588689)

### Purpose:

The main goal is to check that SCAPI meets the requirements. A secondary goal is to find bugs and undesired behavior. We aim to detect software failures so that defects may be discovered and corrected. We also want to make sure that new bugs are not introduced in new versions.

### Package:

All testing files will be under package edu.biu.scapi.tests. Sub packages will be dedicated for each tool.

* edu.biu.scapi.tests.comm – testing of the communication layer
* edu.biu.scapi.tests.primitives – testing of the primitive's layer

### Output:

The output of the tests will be written to a file upon request.

### Timing:

The tests will be performed every version for regression tests. The testing package is a standalone application. To keep track, every version will include the test result files produced by the test package.

### Checking results:

* If all the results for proper behavior pass. The outcome of the test is success.
* The results will be given in files. We will automatically check if there is any difference between the results of the current version and a past tested version. The automatic check will output the difference in a new file.

## Communication layer.

### Proper and simple behavior:

The first test will be to check that the package is working as expected. In this test there will be 2 or 3 parties with different IP addresses and a couple of messages passed between them. This simple test will be composed of several tests which include different configurations. We will use the following configurations.

1. Use pre-defined keys and do not ask for the key exchange algorithm.
2. Key exchange – Ask for the different key exchange algorithms.
3. Naïve success function.
4. At first stage, use the TwoPartiesSuccess success algorithm. After the rest of the success functions will be coded, check them too.
5. Plain channel
6. Encrypted Channel
7. Authenticated channel
8. Secure (both) channel
9. As a unit test check different timeouts.

### Wrong behavior:

In addition we will also provide tests for incorrect behavior and how the tool handles them. The test will print the thrown exceptions. The following tests will be performed.

1. An unauthorized IP/Party will try to connect.
2. One Party will not respond to connection requests and will not initiate connection to the parties it should connect to.
3. Close a socket in the middle of communication setup and after setup.
4. Close the program of one party in the middle of the setup stage and after setup.
5. Overload one station with lots of applications running taking the time of the CPU. Alternatively, use a slow computer. During setup and if setup has succeeded also after setup.
6. Put a lot of traffic on the network in order to cause a major slow down in communication for our tool. During setup and if setup has succeeded also after setup.
7. Make a listening port unavailable.
8. Cause one party (if possible in the success function) to output true and other parties to output false on the success and see how the communication is progressing.
9. Use a key that did not match the session key provided by the key exchange algorithm.
10. Ask for secure channel but do not give a key exchange protocol or a key.
11. Give different list of participants to different parties.
12. Give the same party twice in the list.
13. Give the same IP and port for different parties.
14. Give certain ID's for one party and different ID's to other parties. The IP's will be the same.

### Input size:

Messages of different sizes will be sent. The aim is to test the ability of the tool to handle a variety of message sizes. Specifically, we will use the communication layer to send the following messages.

* One byte
* 10's to 1000's of bytes.
* A message of one megabyte
* A full time movie

The time it takes for such messages to pass will be measured.

### Number of participants:

We want the testing to include numerous parties. This test is not a regression test and will not be performed for every version, rather periodically.

We will test the tool for 10, 100 and 1000 parties. The test will be done in two ways:

* The tests will be maintained in our lab and thus parties will share the same IP's but own different ports.

### To make the tests more realistic, we will use PlanetLab (or any other test bed that we find suitable). PlanetLab is a collection of machines distributed over the globe that can be used. The idea is to place our software on remote computers and check communication.

# The time it takes for the communication layer to set up for the different number of parties will be measured.

### Performance:

We will check the performance of the communication layer. We will measure the amount of time it takes an average size message to get from one point to another. We will also check different environments. Both the proper and wrong behavior tests with varying message sizes will be timed. The overhead of encrypted and authenticated channels will also be compared to simple plain channel. Different number of participants will also be compared.

### Nagle – unit test

The option of nagle is by default turned on in Java. We will run performance test with both options, that is, nagle turned on and off. We consider letting the user determine this option and supplying an API accordingly.

### General design

The tests of the communication layer will be divided into two, regression tests and tests that we do not intend to run for every version (such as causing slow communication, communication traffic, different lists of ip's and other tests that do not include code). The regression test will be part of the design of the testing package. For more information see section [Testing general design](#_General_design:).

## Primitive layer.

### Sources:

SCAPI primitive layer will be tested with the test vectors of NIST and IETF if exist. If we do not find test vectors from NIST or IETF we will look for a different source or make up our own test vectors.

### Wrong input:

In addition we will also provide tests for incorrect arguments and how the tool handles them.

1. Wrong key size
2. Wrong type of key. For example, an RSA key will be passed to the Rabin algorithm.
3. Wrong encoding. Pass a key encoded in one way but indicates an encoding of a different way.
4. Do not call Init. Call functions of algorithms that should be initialized without initializing them.
5. Wrong block size. Call functions of an algorithm that expects a certain size of input with a different size.
6. Size not specified. Call an algorithm that expects the size as argument without specifying the size. For example, call computeBlock of prf with varying input without size.
7. Wrong offset. Pass an array and an offset. Set the offset to be higher than the size of the input array.
8. Wrong argument. For example, an AlgorithmParametersSpec for RSA will be sent to Hmac.
9. Wrong algorithm. Call the prfFactory with "SHA1" as the algorithm name.
10. Wrong algorithm. Call a factory with a name of algorithm that does not exist.
11. Perform bad casting. Cast type that is determined in run time to a wrong type.
12. Pass wrong type of arguments to the discrete log group. For example pass an EC group element to Zp group and also wrong GroupDesc.

### Platforms:

We will check different operating systems for the use of native external libraries. We will match the dll/lib application to the tested environment.

### Performance

There will also be performance tests. We will compare similar algorithms from different sources. The decision which default implementation to supply will take into account the results of the performance tests.

## Testing general design

In general, we will have an interface Test (with a function runTest) that all tests must implement. We will also have a RegressionTest class that will hold a vector of Tests and will perform runTest on each instance in the vector. Also, each primitive interface will have an abstract class. For example, prfTest, prgTest and so on. The concrete tests will derive from these abstract classes that implement the Test interface.

Below is a class diagram containing the Test interface and the abstract classes. It also contains the commTest which is not abstract since we only have one communication feature.



The following class diagram is an example of the derived classes of the PrfTest.



## Appendix

### Table of algorithms/ sources:

In the following table we list all the algorithms we need to test and the source of the vector tests.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Interface | Primitive | NIST | IETF | Other |
| HashTest | SHA1 | <http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf> page 25 |  | Handbook of Applied ryptography, page 345 |
| SHA256 | <http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf> page 33 |  |  |
| SHA224 |  | http://tools.ietf.org/rfc/rfc3874.txt |  |
| SHA512 | <http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf> page 51 |  |  |
| SHA384 | <http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf> page 56 |  |  |
| Universal one-way/target collision-resistant hashing |  |  |  |
| Perfect universal hash functions |  |  |  |
| PRPTest | AES | <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>, page 35 |  |  |
| TripleDes | <http://csrc.nist.gov/publications/nistpubs/800-20/800-20.pdf> for different modes of operation. |  |  |
| PRP varying input/output length |  |  |  |
| PRFTest | Hmac |  | <http://datatracker.ietf.org/doc/rfc4231> |  |
| PRF varying input and output length |  |  |  |
| PRGTest | RC4 |  | <http://datatracker.ietf.org/doc/draft-josefsson-rc4-test-vectors> | http://www.freemedialibrary.com/index.php/RC4\_test\_vectors |
| SHA-based PRG |  |  |  |
| TrapdoorPermutationTest | RSA function |  |  | <http://www.rsa.com/rsalabs/node.asp?id=2125> check the zip file : [RSA-OAEP and RSA-PSS test vectors (.zip file)](ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1-vec.zip) |
| Rabin function |  |  |  |
| DiscreteLogTest | Zp\* |  |  |  |
| EC over the field GF(2m) |  |  |  |
| EC over the field Zp\* |  |  |  |
| KDFTest | HKDF |  |  |  |
| KDFISO18033 |  |  |  |