Garbled Boolean Circuit

Garbled Circuit refers the case of a circuit which we don’t want the evaluator to know the values in the middle of the circuit evaluation. Meaning, each gate's output should tell nothing about its actual result. In order to get this goal we garble the circuit. In case of a Boolean circuit, each wire gets two garbled values (which we called "keys"). During the circuit computation, each gate outputs one of its output wire's garbled values. At the end, the circuit outputs garbled values for each output wire. In order to translate it to a meaningful result, one should call the translate function of the circuit.

Garbled Circuit is a general name for some concrete types of garbled circuits. We implemented the following types:

1. Standard circuit, where all gates have a regular garbled table and all garbled values are sampled randomly independently in each other.
2. Free XOR circuit, where XOR and XORNOT gates have more efficient implementation and both garbled values of each wire are XOR of each other and a fixed delta.
3. Minimize AES setKey operations circuit, a standard circuit using the AES encryption with minimum setKey calls.

Each circuit type can use the row reduction technique or not.

# High level description

We tried to make our implementation as much clear and simple as we can.

There are fixed steps in order to create and use SCAPI's circuit, all of them will be explained later in this document:

* Create a Boolean circuit.
* Create an input object.
* Create a basic concrete circuit instance using the created input object.
* If you wish to use an extended circuit, (For example circuit that accepts garbled values from the user, as will be explained later) you should create the extended circuit using the created basic circuit.
* Now you have a functional garbled circuit. Call which circuit's function you want.

# Detailed description

In this section we will explain each step in the circuit creation.

## Create a Boolean Circuit

Some of the circuit implementations should accept in the constructor an instance of a Boolean circuit. The best way to create such circuit is using a file. The format of the circuit file should be as follows.

1. Number of gates
2. Number of parties
3. For each party:

      Party number

      Number of inputs for that party

      A list of integer labels of each of these input wires.

1. Number of output wires
2. List of integer labels of each of these output wires.
3. For each gate:

      Number of input wires

      Number of output wires

   Input wires labels

   Output Wires labels

     Truth table (as a 0-1 string).

An example file:

1     // One gate

2     // Two parties

1 1 1 // Party one, has one input wire, labeled "1"

2 1 2 // Party two, has one input wire, labeled "2"

1     // One output wire

3 // The output wire, labeled "3"

2 1 1 2 3 0001 // The first (and only) gate has 2 input wire labeled "1" and "2", one output // wire labeled "3" and the truth table is 0001 (AND gate).

## Input classes

Some of the circuit concrete classes should accept an input object.

The given input object will actually determine which type of circuit (Standard, FreeXOR or MinimizeAES) the user want to use and if the row reduction technique should be used or not.

For example, when creating a StandardCircuitInput, the created circuit will be a Standard Garbled Circuit, meaning it uses the representation of garbled table for each gate. Creating a FreeXorCircuitInput will make the created circuit to be a FreeXOR Circuit, meaning all XOR and XORNOT gates will be optimized.

Each input object has a constructor that accepts, along with the necessary parameters for the circuit, a Boolean indicates if the row reduction technique should be used or not.

In order to use row reduction, a KDF object is needed. For that reason, there is a setKDF function that accepts a kdf.

In case the user wants to use the row reduction algorithm, and he didn't call the setKDF function, a default KDF implementation will be used.

In case the user does not want to use the row reduction algorithm, and he did call the setKDF function, an exception will be thrown.

### Code examples

Creating an input object for a Free XOR Circuit without using the row reduction technique:

CircuitInput input = **new** FreeXORCircuitInput(bc, mes, **false**);

Creating an input object for a Free XOR Circuit using the row reduction technique:

CircuitInput input = **new** FreeXORCircuitInput(bc, mes, **true**);

input.setKDF(**new** HKDF(**new** BcHMAC("SHA-256"))); //Optional

## Concrete circuit classes

Garbled Circuit has multiple features and options, like the ability to accept some of the garbled values (which we call "key" in the code), the ability to "hash" the circuit, etc.

In order to be as clearer as possible we decided to have two circuit types:

1. A "basic" circuit for the ordinary user or protocol that has no extra features.
2. An extended circuit that has all the above mentioned features. This circuit will usually be used in malicious protocols.

Each type will now be documented and explained.

### The basic garbled circuit is GarbledBooleanCircuit

#### Functionalities

The basic Garbled Circuit is built from a given Boolean circuit and has the following main functions:

1. **public** CircuitCreationValues garble():

Garble all wire values and build a garbled table for each gate. It returns the input and output garbled values and translation table.

1. **public** CircuitCreationValues garble(**byte**[] seed)

Garble all wire values using the given seed. Then, build a garbled table for each gate. As the previous garble function, it returns the input and output garbled values and translation table.

1. **public** void setGarbledInputFromUngarbledInput(Map<Integer, Byte> ungarbledInput, Map<Integer, SecretKey[]> allInputWireValues)and

**public** void setInputs(Map<Integer, GarbledWire> presetInputWires):

Giving a boolean input, translate it to a garbled input and set it.

1. **public** HashMap<Integer, GarbledWire> compute():

Compute the circuit, meaning call each gate with the current input in a topological order until the last gates output the circuit's output.

1. **public** boolean verify(Map<Integer, SecretKey[]> allInputWireValues):

Giving both garbled values of each input wire, verify that the current circuit's garbled tables are indeed the garbling from the given input garbled values and that the current translation table is valid.

1. **public** **boolean** internalVerify(Map<Integer, SecretKey[]> allInputWireValues, Map<Integer, SecretKey[]> allOutputWireValues):

Giving both garbled values of each input wire, verify that the current circuit's garbled tables is indeed the garbling from the given input garbled values.

In contrary to the previous function, this function does not check the translation table, but instead set the generated output keys to allOutputWireValues.

This function is used, for example, in cases when the circuit is part of a bigger circuit. Thus, it should not have a translation table in order to not expose any information in the middle of the computation.

1. **public** Map<Integer, Wire> translate(Map<Integer, GarbledWire> garbledOutput):

Giving the output from the compute function, translate it to a meaningful Boolean output.

1. **public** Map<Integer, Wire> verifiedTranslate(Map<Integer, GarbledWire> garbledOutput, Map<Integer, SecretKey[]> allOutputWireValues):

Giving the output from the compute function and both garbled values of each output wire, verify that the output from the compute function is valid (for each wire, the output is one of the two possible values) and translate it to a meaningful Boolean output.

#### Usage

The concrete class that implements this circuit is GarbledBooleanCircuitImp.

Steps to create a GarbledBooleanCircuitImp:

1. Create a BooleanCircuit using a file.
2. Create a CircuitInput object.
3. Create a GarbledBooleanCircuitImp using the created input object.
4. Once you have a garbledBooleanCircuit instance you can use all GarbledCircuit's functionalities.

#### Code examples

1. Creating a standard circuit, without using row reduction:

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AESCircuit.txt"));

MultiKeyEncryptionScheme mes = **new** HashingMultiKeyEncryption();

//Create an input object contains the boolean circuit and the encryption scheme.

StandardCircuitInput input = **new** StandardCircuitInput(bc, mes, **new** SecureRandom(), **false**);

//Create a garbled boolean circuit using the given input object. GarbledBooleanCircuit gbc = **new** GarbledBooleanCircuitImp(input);

While you have the circuit instance, the next steps are identical for all extended circuits:

//Garble the circuit.

CircuitCreationValues wireValues = gbc.garble();

//Set the inputs of the circuit (given userInputs, inputs from the user).

gbc.setGarbledInputFromUngarbledInput(userInputs, wireValues.getAllInputWireValues());

gbc.verify(wireValues.getAllInputWireValues());

//Compute the circuit.

Map<Integer, GarbledWire> garbledOutput = **null**;

**try** {

garbledOutput = gbc.compute();

} **catch** (NotAllInputsSetException e) {

// Should not occur since all inputs have been set.

}

//Translate the circuit.

Map<Integer, Wire> output = gbc.verifiedTranslate(garbledOutput, wireValues.getAllOutputWireValues());

1. Creating a MinimizeAESSetKey circuit, using row reduction:

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AESCircuit.txt"));

AES aes = **new** CryptoPpAES();

MultiKeyEncryptionScheme mes = **new** AES128MultiKeyEncryption(aes);

//Create an input object contains the boolean circuit and the encryption scheme.

MinimizeAESSetKeyCircuitInput input = **new** MinimizeAESSetKeyCircuitInput(bc, aes, **new** SecureRandom(), **true**);

Now one can continue as described above, create the GarbledBooleanCircuit usind the input object and use it.

### GarbledBooleanCircuitExtended

The garbledBooleanCicruit extended is an extended circuit that adds extra features to the basic circuit. It is used mostly and designed for malicious protocols.

The extended circuit holds an instance of basic circuit and calsl its functionality when necessary.

#### Functionalities

In addition to the basic circuit abilities, this circuit added the following features:

* The ability to set the keys (which are the garbled values) of the input or/and output wires. The idea is if the user set input keys, then the circuit adds input identity gates for each input wire. When garbling, the internal circuit sample its keys as usual, while the extended circuit, using the added identity gates, maps the given keys to the sampled ones.

The output wires case is identical.

The functions to set the keys are:

1. **public** **void** setInputKeys(Map<Integer, SecretKey[]> inputValues):

This function sets the given keys as the input garbled values.

1. **public** **void** setOutputKeys(Map<Integer, SecretKey[]> outputValues):

This function sets the given garbled values as the output garbled values.

* The ability to "hash" the circuit. Meaning, compute a hash function on the circuit's garbled table and translation table. There are two function regarding the hash:

1. **public** **byte**[] getHashedCircuit(CryptographicHash hash):

This function returns the result of the hash function on the circuit's garbled table and translation table.

1. **public** **boolean** verifyHashedCircuit(CryptographicHash hash, **byte**[] hashedCircuit):

This function verifies that the given hashedCircuit is indeed the result of the given hash on the circuit's garbled tables and translation table.

* Extra verify functions:

1. **public** **boolean** verify(Map<Integer, SecretKey[]> allInputWireValues, Map<Integer, SecretKey[]> allOutputWireValues):

Giving both garbled values of each input wire, verify that the current circuit's garbled tables are indeed the garbling from the given input garbled values and that the current translation table is valid. It also verifies that the resulted output keys are equal to the given allOutputWireValues.

1. **public** **boolean** verify(**byte**[] seed, Map<Integer, SecretKey[]> allInputGarbledValues, Map<Integer, SecretKey[]> allOutputGarbledValues, CryptographicHash hash, **byte**[] hashedCircuit):

This verify function samples the keys using the given seed, then computes the hash function on the circuit's garbled tables and translation table. Then, it checks that the result is equal to the given hashedCircuit.

#### Usage

The concrete class that implements this circuit is GarbledBooleanCircuitExtendedImp.

Steps for create a GarbledBooleanCircuitExtendedImp:

1. Create a BooleanCircuit using a file.
2. Create a CircuitInput object.
3. Create a GarbledBooleanCircuitImp using the created input object.
4. Create a GarbledBooleanCircuitExtendedImp using the created garbled circuit.
5. Once you have a GarbledBooleanCircuitExtended instance you can use all the above functionalities.

#### Code examples

1. Creating a Free XOR circuit, without using row reduction:

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AES\_Final-2.txt"));

MultiKeyEncryptionScheme mes = **new** AESFixedKeyMultiKeyEncryption();

//Create an input object contains the boolean circuit and the encryption scheme.

FreeXORCircuitInput input = **new** FreeXORCircuitInput(bc, mes, **false**);

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuit gbc = **new** GarbledBooleanCircuitImp(input);

GarbledBooleanCircuitExtendedImp circuit = **new** GarbledBooleanCircuitExtendedImp(gbc, mes);

While you have the circuit instance, the next steps are identical for all extended circuits:

//Fix the input wires' values, in this example we sample the keys here.

//Put the fixed values in a HashMap using the wire label as the key and both values as the value.

Map<Integer, SecretKey[]> inputWiresValues = **new** HashMap<Integer, SecretKey[]>();

ArrayList<Integer> inputLabels = **new** ArrayList<Integer>();

**try** {

inputLabels.addAll(input.getUngarbledCircuit().getInputWireIndices(1));

inputLabels.addAll(input.getUngarbledCircuit().getInputWireIndices(2));

} **catch** (NoSuchPartyException e2) {

// Should not occur since the given parties' numbers are correct.

}

**for** (**int** i=0; i<inputLabels.size(); i++){

SecretKey[] keys = **new** SecretKey[2];

keys[0] = mes.generateKey();

keys[1] = mes.generateKey();

inputWiresValues.put(inputLabels.get(i), keys);

}

//Fix the output wires' values, in this example we sample the keys here.

.

//Put the fixed values in a HashMap using the wire label as the key and both values as the value.

Map<Integer, SecretKey[]> outputWiresValues = **new** HashMap<Integer, SecretKey[]>();

**int**[] outputLabels = input.getUngarbledCircuit().getOutputWireIndices();

**for** (**int** i=0; i<outputLabels.length; i++){

SecretKey[] keys = **new** SecretKey[2];

keys[0] = mes.generateKey();

keys[1] = mes.generateKey();

outputWiresValues.put(outputLabels[i], keys);

}

//Garble the circuit.

circuit.setInputKeys(inputWiresValues);

circuit.setOutputKeys(outputWiresValues);

CircuitCreationValues wireValues = circuit.garble();

//Set the inputs of the circuit.

gbc.setGarbledInputFromUngarbledInput(userInputs, wireValues.getAllInputWireValues());

circuit.verify(wireValues.getAllInputWireValues());

//Compute the circuit.

Map<Integer, GarbledWire> garbledOutput = **null**;

**try** {

garbledOutput = circuit.compute();

} **catch** (NotAllInputsSetException e) {

// Should not occur since all inputs have been set.

}

//Translate the circuit.

circuit.verifiedTranslate(garbledOutput, wireValues.getAllOutputWireValues());

1. Creating a standard circuit, using row reduction and a seed:

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AESCircuit.txt"));

MultiKeyEncryptionScheme mes = **new** HashingMultiKeyEncryption();

//Create an input object contains the boolean circuit and the encryption scheme.

StandardCircuitInput input = **new** StandardCircuitInput(bc, mes, **new** SecureRandom(), **true**);

**byte**[] seed = **new** **byte**[]{1,2,3,4};

CryptographicHash hash = **new** BcSHA1();

PseudorandomGenerator prg = **new** BcRC4();

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuitImp gbc = **new** GarbledBooleanCircuitImp(input, prg);

GarbledBooleanCircuitExtendedImp circuit = **new** GarbledBooleanCircuitExtendedImp(gbc, mes, prg);

//Fix the input wires' values, in this example we sample the keys here.

//Put the fixed values in a HashMap using the wire label as the key and both values as the value.

Map<Integer, SecretKey[]> inputWiresValues = **new** HashMap<Integer, SecretKey[]>();

ArrayList<Integer> inputLabels = **new** ArrayList<Integer>();

**try** {

inputLabels.addAll(input.getUngarbledCircuit().getInputWireIndices(1));

inputLabels.addAll(input.getUngarbledCircuit().getInputWireIndices(2));

} **catch** (NoSuchPartyException e2) {

// Should not occur since the given parties' numbers are correct.

}

**for** (**int** i=0; i<inputLabels.size(); i++){

SecretKey[] keys = **new** SecretKey[2];

keys[0] = mes.generateKey();

keys[1] = mes.generateKey();

inputWiresValues.put(inputLabels.get(i), keys);

}

//Garble the circuit.

circuit.setInputKeys(inputWiresValues);

CircuitCreationValues wireValues = **null**;

**try** {

wireValues = circuit.garble(seed);

} **catch** (InvalidKeyException e2) {

// **TODO** Auto-generated catch block

e2.printStackTrace();

}

//Set the inputs of the circuit.

gbc.setGarbledInputFromUngarbledInput(userInputs, wireValues.getAllInputWireValues());

//Verify the circuit.

**try** {

circuit.verify(seed, wireValues.getAllInputWireValues(), **null**, hash, circuit.getHashedCircuit(hash));

} **catch** (InvalidKeyException e1) {

// Should not occur since the given seed is valid for the given prg.

}

//Compute the circuit.

Map<Integer, GarbledWire> garbledOutput = **null**;

**try** {

garbledOutput = circuit.compute();

} **catch** (NotAllInputsSetException e) {

// Should not occur since all inputs have been set.

}

//Translate the circuit.

circuit.verifiedTranslate(garbledOutput, wireValues.getAllOutputWireValues());

# Examples – create a circuit step by step

As we said before there are few steps one should follow in order to get a functioning circuit object:

1. Create an input object.
2. Create a circuit object.
3. Create an extended circuit object – optional.

Below each step will be exampled.

## Create an input object

### Examples of creating input classes for each circuit type, without using row reduction:

Note that all input classes need a Boolean circuit and a MultyKeyEncryption scheme.

* Standard circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AESCircuit.txt"));

MultiKeyEncryptionScheme mes = **new** HashingMultiKeyEncryption();

//Create an input object contains the boolean circuit and the encryption scheme.

StandardCircuitInput input = **new** StandardCircuitInput(bc, mes, **new** SecureRandom(), **false**);

* Free XOR circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AES\_Final-2.txt"));

MultiKeyEncryptionScheme mes = **new** AESFixedKeyMultiKeyEncryption();

//Create an input object contains the boolean circuit and the encryption scheme.

FreeXORCircuitInput input = **new** FreeXORCircuitInput(bc, mes, **false**);

* Minimize AES set Key circuit:

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AESCircuit.txt"));

AES aes = **new** CryptoPpAES();

MultiKeyEncryptionScheme mes = **new** AES128MultiKeyEncryption(aes);

//Create an input object contains the boolean circuit and the encryption scheme.

MinimizeAESSetKeyCircuitInput input = **new** MinimizeAESSetKeyCircuitInput(bc, aes, **new** SecureRandom(), **false**);

### Examples of creating input classes for each circuit type, using row reduction:

Note that all input classes need a Boolean circuit and a MultyKeyEncryption scheme.

* Standard circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AESCircuit.txt"));

MultiKeyEncryptionScheme mes = **new** HashingMultiKeyEncryption();

//Create an input object contains the boolean circuit and the encryption scheme.

StandardCircuitInput input = **new** StandardCircuitInput(bc, mes, **new** SecureRandom(), **true**);

* Free XOR circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AES\_Final-2.txt"));

MultiKeyEncryptionScheme mes = **new** AESFixedKeyMultiKeyEncryption();

//Create an input object contains the boolean circuit and the encryption scheme.

FreeXORCircuitInput input = **new** FreeXORCircuitInput(bc, mes, **true**);

* Minimize AES set Key circuit:

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AESCircuit.txt"));

AES aes = **new** CryptoPpAES();

MultiKeyEncryptionScheme mes = **new** AES128MultiKeyEncryption(aes);

//Create an input object contains the boolean circuit and the encryption scheme.

MinimizeAESSetKeyCircuitInput input = **new** MinimizeAESSetKeyCircuitInput(bc, aes, **new** SecureRandom(), **true**);

## Create a circuit object using a given input instance

### Create a circuit that uses a MultiKeyEncryption scheme for key genearation:

The input object should be one of the above exampled objects.

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuit gbc = **new** GarbledBooleanCircuitImp(input);

### Create a circuit that uses a seed for key generation:

The input object should be one of the above exampled objects.

**byte**[] seed = **new** **byte**[]{1,2,3,4};

PseudorandomGenerator prg = **new** BcRC4();

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuitImp gbc = **new** GarbledBooleanCircuitImp(input, prg);

## Create an extended circuit using a given "basic" circuit:

### Create an extended circuit that uses a MultiKeyEncryption scheme for key genearation:

The given circuit object should be one of the above exampled circuits.

GarbledBooleanCircuitExtendedImp circuit = **new** GarbledBooleanCircuitExtendedImp(gbc, mes);

### Create a circuit that uses a seed for key generation:

The given circuit object should be a "basic" circuit that also uses a seed for key generation.

**byte**[] seed = **new** **byte**[]{1,2,3,4};

CryptographicHash hash = **new** BcSHA1();

PseudorandomGenerator prg = **new** BcRC4(); //Same for the "basic" given

//circuit.

GarbledBooleanCircuitExtendedImp circuit = **new** GarbledBooleanCircuitExtendedImp(gbc, mes, prg);

# Examples – create a whole circuit

## Examples of creating "basic" circuits without row reduction:

### Standard circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AESCircuit.txt"));

MultiKeyEncryptionScheme mes = **new** HashingMultiKeyEncryption();

//Create an input object contains the boolean circuit and the encryption scheme.

StandardCircuitInput input = **new** StandardCircuitInput(bc, mes, **new** SecureRandom(), **false**);

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuit gbc = **new** GarbledBooleanCircuitImp(input);

### Free XOR circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AES\_Final-2.txt"));

MultiKeyEncryptionScheme mes = **new** AESFixedKeyMultiKeyEncryption();

//Create an input object contains the boolean circuit and the encryption scheme.

FreeXORCircuitInput input = **new** FreeXORCircuitInput(bc, mes, **false**);

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuit gbc = **new** GarbledBooleanCircuitImp(input);

### Minimize AES set Key circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AESCircuit.txt"));

AES aes = **new** CryptoPpAES();

MultiKeyEncryptionScheme mes = **new** AES128MultiKeyEncryption(aes);

//Create an input object contains the boolean circuit and the encryption scheme.

MinimizeAESSetKeyCircuitInput input = **new** MinimizeAESSetKeyCircuitInput(bc, aes, **new** SecureRandom(), **false**);

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuit gbc = **new** GarbledBooleanCircuitImp(input);

## Examples of creating "basic" circuits with row reduction:

### Standard circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AESCircuit.txt"));

MultiKeyEncryptionScheme mes = **new** HashingMultiKeyEncryption();

//Create an input object contains the boolean circuit and the encryption scheme.

StandardCircuitInput input = **new** StandardCircuitInput(bc, mes, **new** SecureRandom(), **true**);

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuit gbc = **new** GarbledBooleanCircuitImp(input);

### Free XOR circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AES\_Final-2.txt"));

MultiKeyEncryptionScheme mes = **new** AESFixedKeyMultiKeyEncryption();

//Create an input object contains the boolean circuit and the encryption scheme.

FreeXORCircuitInput input = **new** FreeXORCircuitInput(bc, mes, **true**);

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuit gbc = **new** GarbledBooleanCircuitImp(input);

### Minimize AES set Key circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AESCircuit.txt"));

AES aes = **new** CryptoPpAES();

MultiKeyEncryptionScheme mes = **new** AES128MultiKeyEncryption(aes);

//Create an input object contains the boolean circuit and the encryption scheme.

MinimizeAESSetKeyCircuitInput input = **new** MinimizeAESSetKeyCircuitInput(bc, aes, **new** SecureRandom(), **true**);

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuit gbc = **new** GarbledBooleanCircuitImp(input);

## Examples of creating a "basic" circuits without row reduction but using a seed for key generation:

### Standard circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AESCircuit.txt"));

MultiKeyEncryptionScheme mes = **new** HashingMultiKeyEncryption();

//Create an input object contains the boolean circuit and the encryption scheme.

StandardCircuitInput input = **new** StandardCircuitInput(bc, mes, **new** SecureRandom(), **false**);

**byte**[] seed = **new** **byte**[]{1,2,3,4};

PseudorandomGenerator prg = **new** BcRC4();

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuitImp gbc = **new** GarbledBooleanCircuitImp(input, prg);

### Free XOR circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AES\_Final-2.txt"));

MultiKeyEncryptionScheme mes = **new** AESFixedKeyMultiKeyEncryption();

//Create an input object contains the boolean circuit and the encryption scheme.

FreeXORCircuitInput input = **new** FreeXORCircuitInput(bc, mes, **false**);

**byte**[] seed = **new** **byte**[]{1,2,3,4};

PseudorandomGenerator prg = **new** BcRC4();

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuitImp gbc = **new** GarbledBooleanCircuitImp(input, prg);

### Minimize AES set Key circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AESCircuit.txt"));

AES aes = **new** CryptoPpAES();

MultiKeyEncryptionScheme mes = **new** AES128MultiKeyEncryption(aes);

//Create an input object contains the boolean circuit and the encryption scheme.

MinimizeAESSetKeyCircuitInput input = **new** MinimizeAESSetKeyCircuitInput(bc, aes, **new** SecureRandom(), **false**);

**byte**[] seed = **new** **byte**[]{1,2,3,4};

PseudorandomGenerator prg = **new** BcRC4();

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuitImp gbc = **new** GarbledBooleanCircuitImp(input, prg);

## Examples of creating "basic" circuits with row reduction and using a seed for key generation:

### Standard circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AESCircuit.txt"));

MultiKeyEncryptionScheme mes = **new** HashingMultiKeyEncryption();

//Create an input object contains the boolean circuit and the encryption scheme.

StandardCircuitInput input = **new** StandardCircuitInput(bc, mes, **new** SecureRandom(), **true**);

**byte**[] seed = **new** **byte**[]{1,2,3,4};

PseudorandomGenerator prg = **new** BcRC4();

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuitImp gbc = **new** GarbledBooleanCircuitImp(input, prg);

### Free XOR circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AES\_Final-2.txt"));

MultiKeyEncryptionScheme mes = **new** AESFixedKeyMultiKeyEncryption();

//Create an input object contains the boolean circuit and the encryption scheme.

FreeXORCircuitInput input = **new** FreeXORCircuitInput(bc, mes, **true**);

**byte**[] seed = **new** **byte**[]{1,2,3,4};

PseudorandomGenerator prg = **new** BcRC4();

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuitImp gbc = **new** GarbledBooleanCircuitImp(input, prg);

### Minimize AES set Key circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AESCircuit.txt"));

AES aes = **new** CryptoPpAES();

MultiKeyEncryptionScheme mes = **new** AES128MultiKeyEncryption(aes);

//Create an input object contains the boolean circuit and the encryption scheme.

MinimizeAESSetKeyCircuitInput input = **new** MinimizeAESSetKeyCircuitInput(bc, aes, **new** SecureRandom(), **true**);

**byte**[] seed = **new** **byte**[]{1,2,3,4};

PseudorandomGenerator prg = **new** BcRC4();

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuitImp gbc = **new** GarbledBooleanCircuitImp(input, prg);

## Examples of creating extended circuits without row reduction:

### Standard circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AESCircuit.txt"));

MultiKeyEncryptionScheme mes = **new** HashingMultiKeyEncryption();

//Create an input object contains the boolean circuit and the encryption scheme.

StandardCircuitInput input = **new** StandardCircuitInput(bc, mes, **new** SecureRandom(), **false**);

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuit gbc = **new** GarbledBooleanCircuitImp(input);

GarbledBooleanCircuitExtendedImp circuit = **new** GarbledBooleanCircuitExtendedImp(gbc, mes);

### Free XOR circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AES\_Final-2.txt"));

MultiKeyEncryptionScheme mes = **new** AESFixedKeyMultiKeyEncryption();

//Create an input object contains the boolean circuit and the encryption scheme.

FreeXORCircuitInput input = **new** FreeXORCircuitInput(bc, mes, **false**);

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuit gbc = **new** GarbledBooleanCircuitImp(input);

GarbledBooleanCircuitExtendedImp circuit = **new** GarbledBooleanCircuitExtendedImp(gbc, mes);

### Minimize AES set Key circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AESCircuit.txt"));

AES aes = **new** CryptoPpAES();

MultiKeyEncryptionScheme mes = **new** AES128MultiKeyEncryption(aes);

//Create an input object contains the boolean circuit and the encryption scheme.

MinimizeAESSetKeyCircuitInput input = **new** MinimizeAESSetKeyCircuitInput(bc, aes, **new** SecureRandom(), **false**);

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuit gbc = **new** GarbledBooleanCircuitImp(input);

GarbledBooleanCircuitExtendedImp circuit = **new** GarbledBooleanCircuitExtendedImp(gbc, mes);

## Examples of creating extended circuits with row reduction:

### Standard circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AESCircuit.txt"));

MultiKeyEncryptionScheme mes = **new** HashingMultiKeyEncryption();

//Create an input object contains the boolean circuit and the encryption scheme.

StandardCircuitInput input = **new** StandardCircuitInput(bc, mes, **new** SecureRandom(), **true**);

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuit gbc = **new** GarbledBooleanCircuitImp(input);

GarbledBooleanCircuitExtendedImp circuit = **new** GarbledBooleanCircuitExtendedImp(gbc, mes);

### Free XOR circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AES\_Final-2.txt"));

MultiKeyEncryptionScheme mes = **new** AESFixedKeyMultiKeyEncryption();

//Create an input object contains the boolean circuit and the encryption scheme.

FreeXORCircuitInput input = **new** FreeXORCircuitInput(bc, mes, **true**);

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuit gbc = **new** GarbledBooleanCircuitImp(input);

GarbledBooleanCircuitExtendedImp circuit = **new** GarbledBooleanCircuitExtendedImp(gbc, mes);

### Minimize AES set Key circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AESCircuit.txt"));

AES aes = **new** CryptoPpAES();

MultiKeyEncryptionScheme mes = **new** AES128MultiKeyEncryption(aes);

//Create an input object contains the boolean circuit and the encryption scheme.

MinimizeAESSetKeyCircuitInput input = **new** MinimizeAESSetKeyCircuitInput(bc, aes, **new** SecureRandom(), **true**);

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuit gbc = **new** GarbledBooleanCircuitImp(input);

GarbledBooleanCircuitExtendedImp circuit = **new** GarbledBooleanCircuitExtendedImp(gbc, mes);

## Examples of creating extended circuits without row reduction but using a seed for key generation:

### Standard circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AESCircuit.txt"));

MultiKeyEncryptionScheme mes = **new** HashingMultiKeyEncryption();

//Create an input object contains the boolean circuit and the encryption scheme.

StandardCircuitInput input = **new** StandardCircuitInput(bc, mes, **new** SecureRandom(), **false**);

**byte**[] seed = **new** **byte**[]{1,2,3,4};

PseudorandomGenerator prg = **new** BcRC4();

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuitImp gbc = **new** GarbledBooleanCircuitImp(input, prg);

GarbledBooleanCircuitExtendedImp circuit = **new** GarbledBooleanCircuitExtendedImp(gbc, mes, prg);

### Free XOR circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AES\_Final-2.txt"));

MultiKeyEncryptionScheme mes = **new** AESFixedKeyMultiKeyEncryption();

//Create an input object contains the boolean circuit and the encryption scheme.

FreeXORCircuitInput input = **new** FreeXORCircuitInput(bc, mes, **false**);

**byte**[] seed = **new** **byte**[]{1,2,3,4};

PseudorandomGenerator prg = **new** BcRC4();

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuitImp gbc = **new** GarbledBooleanCircuitImp(input, prg);

GarbledBooleanCircuitExtendedImp circuit = **new** GarbledBooleanCircuitExtendedImp(gbc, mes, prg);

### Minimize AES set Key circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AESCircuit.txt"));

AES aes = **new** CryptoPpAES();

MultiKeyEncryptionScheme mes = **new** AES128MultiKeyEncryption(aes);

//Create an input object contains the boolean circuit and the encryption scheme.

MinimizeAESSetKeyCircuitInput input = **new** MinimizeAESSetKeyCircuitInput(bc, aes, **new** SecureRandom(), **false**);

**byte**[] seed = **new** **byte**[]{1,2,3,4};

PseudorandomGenerator prg = **new** BcRC4();

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuitImp gbc = **new** GarbledBooleanCircuitImp(input, prg);

GarbledBooleanCircuitExtendedImp circuit = **new** GarbledBooleanCircuitExtendedImp(gbc, mes, prg);

## Examples of creating extended circuits with row reduction and using a seed for key generation:

### Standard circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AESCircuit.txt"));

MultiKeyEncryptionScheme mes = **new** HashingMultiKeyEncryption();

//Create an input object contains the boolean circuit and the encryption scheme.

StandardCircuitInput input = **new** StandardCircuitInput(bc, mes, **new** SecureRandom(), **true**);

**byte**[] seed = **new** **byte**[]{1,2,3,4};

PseudorandomGenerator prg = **new** BcRC4();

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuitImp gbc = **new** GarbledBooleanCircuitImp(input, prg);

GarbledBooleanCircuitExtendedImp circuit = **new** GarbledBooleanCircuitExtendedImp(gbc, mes, prg);

### Free XOR circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AES\_Final-2.txt"));

MultiKeyEncryptionScheme mes = **new** AESFixedKeyMultiKeyEncryption();

//Create an input object contains the boolean circuit and the encryption scheme.

FreeXORCircuitInput input = **new** FreeXORCircuitInput(bc, mes, **true**);

**byte**[] seed = **new** **byte**[]{1,2,3,4};

PseudorandomGenerator prg = **new** BcRC4();

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuitImp gbc = **new** GarbledBooleanCircuitImp(input, prg);

GarbledBooleanCircuitExtendedImp circuit = **new** GarbledBooleanCircuitExtendedImp(gbc, mes, prg);

### Minimize AES set Key circuit

//Create a boolean circuit. Read the circuit's parameters from a file.

BooleanCircuit bc = **new** BooleanCircuit(**new** File("AESCircuit.txt"));

AES aes = **new** CryptoPpAES();

MultiKeyEncryptionScheme mes = **new** AES128MultiKeyEncryption(aes);

//Create an input object contains the boolean circuit and the encryption scheme.

MinimizeAESSetKeyCircuitInput input = **new** MinimizeAESSetKeyCircuitInput(bc, aes, **new** SecureRandom(), **true**);

**byte**[] seed = **new** **byte**[]{1,2,3,4};

PseudorandomGenerator prg = **new** BcRC4();

//Create a garbled boolean circuit using the given input object.

GarbledBooleanCircuitImp gbc = **new** GarbledBooleanCircuitImp(input, prg);

GarbledBooleanCircuitExtendedImp circuit = **new** GarbledBooleanCircuitExtendedImp(gbc, mes, prg);

# Examples – circuit functionality

Once you have a garbled circuit you can use the circuit's functions:

## Garbling using a MultiKeyEncryptionScheme

Given a GarbledBooleanCircuit and inputs for the circuit (userInputs):

//Garble the circuit.

CircuitCreationValues wireValues = gbc.garble();

//Set the inputs of the circuit.

gbc.setGarbledInputFromUngarbledInput(userInputs, wireValues.getAllInputWireValues());

//Verify the circuit.

gbc.verify(wireValues.getAllInputWireValues());

//Compute the circuit.

Map<Integer, GarbledWire> garbledOutput = **null**;

**try** {

garbledOutput = gbc.compute();

} **catch** (NotAllInputsSetException e) {

// Should not occur since all inputs have been set.

}

//Translate the circuit.

Map<Integer, Wire> output = gbc.verifiedTranslate(garbledOutput, wireValues.getAllOutputWireValues());

## Garbling using a seed

Given a GarbledBooleanCircuit, a seed and inputs for the circuit (userInputs):

//Garble the circuit.

CircuitCreationValues wireValues = **null**;

**try** {

wireValues = gbc.garble(seed);

} **catch** (InvalidKeyException e2) {

// Should not occur since the given seed is valid for the given prg.

}

//Set the inputs.

gbc.setGarbledInputFromUngarbledInput(userInputs, wireValues.getAllInputWireValues());

//Verify the circuit.

gbc.verify(wireValues.getAllInputWireValues());

//Compute the circuit.

Map<Integer, GarbledWire> garbledOutput = **null**;

**try** {

garbledOutput = gbc.compute();

} **catch** (NotAllInputsSetException e) {

// Should not occur since all inputs have been set.

}

//Translate the circuit.

Map<Integer, Wire> output = gbc.verifiedTranslate(garbledOutput, wireValues.getAllOutputWireValues());

## Extended circuit – Garbling using a MultiKeyEncryptionScheme

Given a GarbledBooleanCircuitExtended and inputs for the circuit (userInputs):

//Given fixed input keys (inputWiresValues) and fixed output keys (outputWiresValues)

//Garble the circuit.

circuit.setInputKeys(inputWiresValues);

circuit.setOutputKeys(outputWiresValues);

CircuitCreationValues wireValues = circuit.garble();

//Set the inputs of the circuit.

gbc.setGarbledInputFromUngarbledInput(userInputs, wireValues.getAllInputWireValues());

//Verify the circuit.

circuit.verify(wireValues.getAllInputWireValues());

//Compute the circuit.

Map<Integer, GarbledWire> garbledOutput = **null**;

**try** {

garbledOutput = circuit.compute();

} **catch** (NotAllInputsSetException e) {

// Should not occur since all inputs have been set.

}

//Translate the circuit.

Map<Integer, Wire> output = circuit.verifiedTranslate(garbledOutput, wireValues.getAllOutputWireValues());

## Extended circuit – Garbling using a seed

Given a GarbledBooleanCircuitExtended, a seed and inputs for the circuit (userInputs):

//Given fixed input keys (inputWiresValues) and fixed output keys (outputWiresValues)

//Garble the circuit.

circuit.setInputKeys(inputWiresValues);

circuit.setOutputKeys(outputWiresValues);

CircuitCreationValues wireValues = **null**;

**try** {

wireValues = circuit.garble(seed);

} **catch** (InvalidKeyException e2) {

// **TODO** Auto-generated catch block

e2.printStackTrace();

}

//Set the inputs of the circuit.

gbc.setGarbledInputFromUngarbledInput(userInputs, wireValues.getAllInputWireValues());

//Verify the circuit **using the seed**.

**try** {

circuit.verify(seed, wireValues.getAllInputWireValues(),

wireValues.getAllOutputWireValues(), hash, circuit.getHashedCircuit(hash));

} **catch** (InvalidKeyException e1) {

// Should not occur since the given seed is valid for the given prg.

}

//Compute the circuit.

Map<Integer, GarbledWire> garbledOutput = **null**;

**try** {

garbledOutput = circuit.compute();

} **catch** (NotAllInputsSetException e) {

// Should not occur since all inputs have been set.

}

//Translate the circuit.

Map<Integer, Wire> output = circuit.verifiedTranslate(garbledOutput, wireValues.getAllOutputWireValues());

An example for using the circuit in two-party protocol can be shown in the [Yao protocol](http://crypto.biu.ac.il/scapi/examples/ExampleYaoProtocol.pdf) at SCAPI's web site.