**Manual: Expanding the Verifier**

Our verifier for the Verificatum Mix-Net is written in a generic way so that future improvements and developments in the Verificatum can easily be supported by our verifier with minimum work being done. This manual explains exactly the 2 steps which are needed in order to add more types of fields, rings, or groups to be supported by our verifier.

**Adding a New Group**

Currently, the verifier supports two types of groups: multiplicative modulo prime groups, and elliptic curves. For each of these two, there is a unique class which implements the IGroup interface. Therefore, in order to add a new type of group, our first step is to create a new class that implements the IGroup interface. The IGroup interface has all the methods needed for the group and thus once all its method have been implemented, we have a new type of group.

To complete this expansion, we need to create the elements of our new group. Note that this step is optional, as we can always use one of our currently supported type of group elements (e.g. create a new group and define its elements to be elliptic curve points). In order to create a new type of group element, all that needs to be done is implement the IGroupElement interface. This interface has all the methods needed for the group element and thus once all its methods have been implemented we have a new type of group element which we can assign to our new type of group.

When dealing with groups, there is one more step to do, and that is going into the ElementsExtractor class and expanding two of its existing methods. The first one is the "unmarshal" method which takes a string representing a certain group and recovers the represented group. This method needs to be expanded of course so the verifier will know how to unmarshal the new type of group we just added.

The second method is "createGroupElement" which takes a byte array representation of a group element as an input and creates the group element represented by the array. This method is used widely throughout our verifier and therefore it is very important that like in the "unmarshal" method, it will be expanded so the verifier will know how our new group elements are represented as byte arrays and will know how to read and create them from such arrays.

Once these steps are completed (implement the IGroup interface, implement the IGroupElement interface, and expand the 2 methods in the ElementsExtractor class) we have officially added a new type of group to our verifier.

**Adding a New Field/Ring**

The steps for adding a new type of field are almost identical to those for adding a new type of ring. Thus, this section will explain about fields, and at the end there will be clarifications for what to do if we are dealing with rings. So, to add a new type of field, first we need to define what the elements of the field are. Currently, our verifier supports one type of field elements which is "IntegerFieldElement". If our new type of field consists of this kind of elements, then we can skip this step and not define a new type of field element. Otherwise, a new class needs to be created representing our new kind of field element. For the sake of this explanation we will call our new class NewFieldElementType.

Our second and last step would be to implement the interface IField over our new type NewFieldElementType. That is, create a new class which implements the interface IField<NewFieldElementType>. The IField interface has all the methods needed for the new field and thus once all its method have been implemented, we have a new type of field.

If we wish to create a new ring rather than a new field, the same steps have to be done (i.e. create a NewRingElementType or use the currently supported "IntegerRingElement"), and then implement the IRing<NewRingElementType> interface. The IRing interface has all the methods needed for the new ring and thus once all its method have been implemented, we have a new type of ring.

**Creating arrays of our new elements**

The arrays in our verifier are also generic, therefore once a new type of element has been created and we wish to create an array containing elements of this type, all we need to do is type:

ArrayOfElements<OurNewTypeOfElement> arrayName = new ArrayOfElements<OurNewTypeOfElement>();