# VALIDATION AND APPROVAL OF PN CIPHERED SUBNETS: SECURITY IN PN SHARING AND STORAGE USING PNML, XMLSECURITY AND XMLSIGNATURE

**Iñigo León Samaniego [(a)], Mercedes Perez-Parte [(b)], Juan-Ignacio Latorre-Biel [(c)], Emilio Jiménez-Macías [(d)]**

[(a,d)] University of La Rioja. Department of Electrical Engineering
[(b,c)] University of La Rioja. Department of Mechanical Engineering

[(a)]inigo.leon@gmail.com,[(b)]mercedes.perez@unirioja.es,[(b)]juan-ignacio.latorre@unirioja.es, [(b)] emilio.jimenez@unirioja.es

**ABSTRACT**

In this work I board the study of Petri nets from the point of view of the security. There several goals. First of all, I will take advantage from my previous work in order to complete a subnetting process. Then, the creation of a PNML extension that allows the representation of subnets structures.

One application of this subnetting and PNML representation is the possibility of hiding part of a Petri net, facing a possible distribution, maintaining the privacy of the critical, secret, or complex parts of the system. However this hidden information is not eliminated from the net, but encrypted inside.

Other application explained is the possibility of digital signature of subnets, providing security services to the net and/or subnets.

My contributions to knowledge are:

1. Comprehensive study of subnets, abstracting their internal structure from the exterior by using front-ends. A method to build these subnets from the complete Petri net is explained and analyzed matrixed.
2. PNML has no way to represent subnets, so I approach a possible PNML extension to do it.
3. Subnetting and PNML extension to represents subnets allow to apply several security technics that offers encryption, data integrity, authentication and non repudiation

Keywords: Petri net, Petri subnet, subnetting PNML, XMLEncryption, XMLSignatues

## 1. INTRODUCTION

### 1.1. Background of the research

Petri nets are widespread for modeling many classes of systems, such as manufacturing logistics processes and services, concurrent systems, etc. However, all these nets are described in a comprehensive way and must have the information of the entire net to determine its evolution.

Furthermore, these nets can be modified with no control of integrity or authoring, for example.

### 1.2. Research problem

The problem occurs when somebody doesn't want to describe the whole subnet. Or, maybe, is wanted one part of the process to be only accessible for one specific person or entity.

The first approach to solve this problem is to take two Petri nets:

- one Petri Net with only the public information, extracting the private data. This is an incomplete model of the process
- another Petri Net with the whole information for the interested person or entity.

As you can notice, this is not an efficient way to publish this kind of Petri nets.

Other problem appears when I want to protect parts of the net from undesired modifications or ensure the authoring of some parts (or the whole net).

### 1.3. Justification of the research

It would be interesting to provide security to a Petri net:

- hiding a part of it. This can be useful, for example, distributing a process we want to be secret (León 2011), or simply to be a part of the net to be complex and do not interest handle for any reason (León 2011).
- avoiding not allowed changes in it.
- authenticating it (or a part of it). Useful to ensure who has developed a Petri net or subnet.
- avoiding the possibility of supplant other people in the authority of the Petri net or some of its parts.

So here is my contribution. I have researched the possibilities of hiding a part of a Petri Net so that everybody can access the public information, maintaining the secret of the private data. This private data is accessible only for authorized people. And not only that: I ensure data integrity, authentication and non-repudiation to Petri nets or subnets.

Some authors study the possibilities of Petri nets reduction (Valette 1979; Suzuki and Murata 1983; Fahmy 1990; Druzhinin and Yuditskii 1992; Fahmy 1993; Xia 2011), grouping in one place or transition a subnet, so that what happens on this subnet, is encapsulated in a single point of execution. However, we want to go further by defining parts of the net that are hidden (not clustered) and what are the implications, studied within network properties.

The main objective of this work is to extract parts (subnets) of a Petri net and provide them of wide security (privacy, integrity, authentication and non-repudiation).

### 1.4. Methodology

In order to achieve this goal, I have defined three milestones:

- Extend Petri Nets in order to define subnets, abstracting the internal structure from the rest of the net using front-ends, focusing on hiding information.
- Choose a lossless and extendible representation of this kind of Petri Nets
- Define both hiding and signing methods for this representation

For the first milestone, I complete my previous works for the creation of the theoretical basis for further study of Petri nets in which certain parts are hidden. So we setup a generic framework of definitions and notations that allow us to deepen in the study of the characteristics and properties of Petri nets and their subnets (Murata 1989; Silva 1985).

Also mention work already carried by other researchers in which we rely for our goal (Silva 1993; David and Alla 2010; Jensen and Kristensen 2009; Peterson 1981).

All of this will be necessary to create the framework that allows us to study occultation in Petri nets. We will expand the vision of Petri nets, providing them with greater functionality, such us attachable subnet (León 2013).

The next step in this work is to choose (or define) a flexible representation of Petri nets that allows us to translate the previous extended nets. This representation has to be really extendible and flexible in order to be able to show actual and future characteristics of Petri nets.

I can advance you that the selected representation is the standard PNML and I have to define and extension for it in order to represent subnets that are going to be secured.

Once selected this representation, the last step is the hiding and signing method (digital signature provide integrity, authentication and non-repudiation services). Once more, I bet for standard protocols like XMLEncryption and XMLSignature.

This is a very basic investigation because I extend the very early definitions of Petri nets. Because of it, the results of this work are very probably extensible to any other development whose basis are the classic Petri nets. For example, I am not going to study colored Petri nets, neither timed Petri nets, etc. But it is very easy to see that the results achieved in this thesis can be applied to them with little problems.

## 2. SUBNETTING

Let's take a Petri Net R = <P, T, α, β> where P is the set of places, T is the set of transitions, α is the pre-incidence function and β is the post-incidence function. We define R' = <P' , T' , α', β'> such that P'□P and T'□T , α' and β' are restrictions of α y β over P' × T' (P' and T' are not empty).

Then we have split the original Petri Net in two disjoint subnets, R' and the rest of the net.

The next step is to analyze the inputs and the outputs of R'. For example, starting from the Figure 1, I want to extract R1 and its interface:

$$C = \begin{array}{c} \\ p_1 \\ \vdots \\ p_m \\ p_{m+1} \\ \vdots \\ p_n \end{array} \begin{pmatrix} t_1 & \cdots & t_s & t_{s+1} & \cdots & t_r \\ a_{11} & \cdots & a_{1s} & a_{1(s+1)} & \cdots & a_{1r} \\ \vdots & R' & \vdots & \vdots & HN & \vdots \\ a_{m1} & \cdots & a_{ms} & a_{m(s+1)} & \cdots & a_{mr} \\ \hline a_{(m+1)1} & \cdots & a_{(m+1)s} & a_{(m+1)(s+1)} & \cdots & a_{(m+1)r} \\ \vdots & HT & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{ns} & a_{n(s+1)} & \cdots & a_{nr} \end{pmatrix}$$

Figure 1: Petri Net divided into two subnets

The resultant extracted interface is:

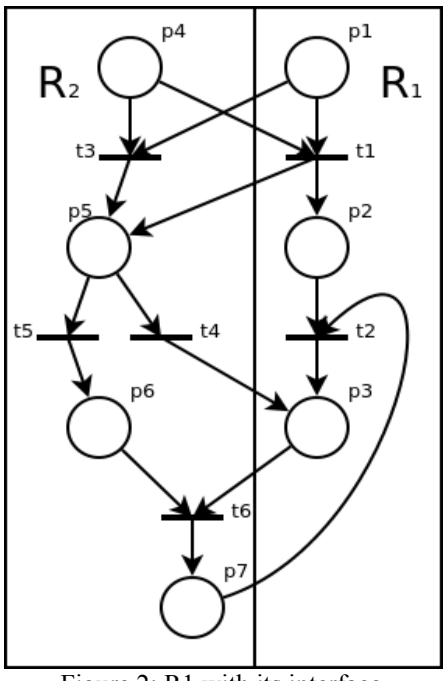
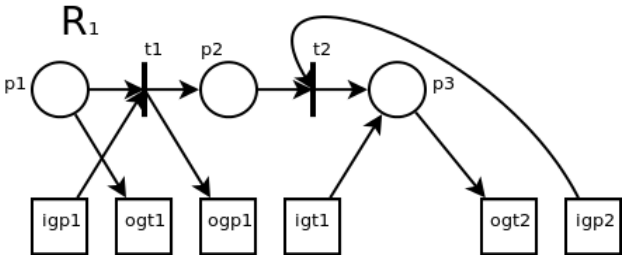



Figure 2: R1 with its interface

So we can define the input interface and the output interface in this way.




or the same:

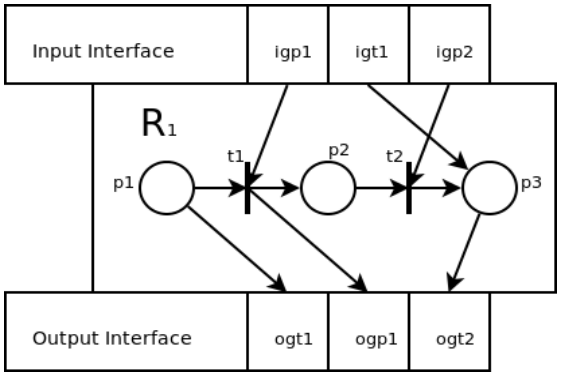



Figure 3: R1 with its input and output interface.

Once this is done, the next step is to implement this information in a Petri Net representation that will allow security implementation. For this objective, I am going to use PNML.

## 3. PETRI NET REPRESENTATION FOR SUBNETS SUPPORT.

### 3.1. Petri net representations

There are four standard ways to represent Petri nets. Each one of them have their properties, advantages and disadvantages. But I want to select one that I am able to represent any kind of Petri net, its subnets and allow to hide information without erasing it.

#### 3.1.1. Graphic representation

This is the clearest and extended way to represent Petri nets. It has a very important advantage and it is that a picture is worth a thousand words.

Subnets can be defined simply drawing a vertical line. The right part is one subnet and the left part is other subnet. Places and transitions can be moved from one location to another depending on the subnet they are situated. This is only an example. Other way would be to use colors for the nodes (same color indicates same subnet) or use rectangles, etc.

So this representation is useful in order to show at one sight the Petri net structure, but I can't choose it for my goals.

I have not been able to discover a way to show some people the hidden information (the hidden subnet). However I will continue using it where a clear idea of the Petri structure if necessary.

#### 3.1.2. Matrix representation

This representation is very useful to study properties and evolution of a Petri net, independently of its graphic representation. We can reorder rows and columns and define subnets in the matrix.

For example, let's take the Petri net represented by the matrix of Figure 4

$$\begin{array}{c|ccc|ccc} & t_1 & t_2 & t_3 & t_4 & t_5 & t_6 \\ p_1 & -1 & 0 & 1 & 0 & 0 & 0 \\ p_2 & -1 & 0 & 0 & 1 & 0 & 1 \\ p_3 & 1 & 0 & 0 & -1 & 0 & 0 \\ p_4 & 1 & -1 & 0 & 2 & 0 & 0 \\ \hline p_5 & 0 & 1 & 0 & 0 & 0 & 0 \\ p_6 & 3 & 0 & -1 & 0 & 0 & 0 \\ p_7 & 0 & 0 & 0 & -1 & -1 & 1 \\ p_8 & 0 & 0 & 0 & 0 & 1 & -1 \end{array}$$

Figure 4: Matrix representation of a Petri net

We could group the places and transitions of the hidden subnet in this way, maintaining those elements other than zero:

$$\begin{array}{c|cc|cc} & \blacksquare & & t_4 & t_5 & t_6 \\ & & & (1,-1,2) & 0 & 1 \\ \hline p_5 & 1 & & 0 & 0 & 0 \\ p_6 & (3,-1) & & 0 & 0 & 0 \\ p_7 & 0 & & -1 & -1 & 1 \\ p_8 & 0 & & 0 & 1 & -1 \end{array}$$

Figure 5: Matrix representation of a Petri net with hidden subnet

With this representation it is possible to study properties and it may be really important as a complement to graphic mode. With both representations together, everyone has a clear idea of the Petri net structure and properties. But I haven't found a way to store information inside the black box.

#### 3.1.3. Equation representation

The third representation way for Petri nets is the equation representation.

Basically, transitions are selected and, for each one, the tokens of the places connected to that transition are modified. This is very useful to compute the evolution of a Petri net, choosing the transition fired.

However it is difficult to find a way for representing subnets with this notation. And, of course, if a subnet cannot be represented, it cannot be hidden.

In the Figure 6 we have an example of equation representation of a Petri net.

```
if (p1>0) then
        p1 <- p1 - 1
        p2 <- p2 + 1
if (p2>0) then
        p2 <- p2 - 1
        p1 <- p1 + 1
        p3 <- p3 + 1
if (p3>0) then
        p3 <- p3 - 1
        p2 <- p2 + 1
        p4 <- p4 + 1
if (p4>0) then
        p4 <- p4 - 1
        p1 <- p1 + 1
        p5 <- p5 + 1
```

Figure 6: Equation representation of a Petri net

I have tried to think about these ways of representation but, in my opinion, no one of them is suitable enough to represent subnets in a clear way than can be occulted. Because of that I have chosen the fourth representation, which is PNML, and is explained in the next section.

#### 3.1.4. PNML Petri Net Marked Language

PNML is a way to represent Petri nets as xml content. The advantages over the other three representation ways described before are clear. By one side, XML is a widely extended format to represent almost everything. In the other side, XML is a robust technology free of errors and it is really flexible. Its flexibility comes from the possibility of adding any kind of labels and functionality with a very little amount of work. Its robustness come from the strict specification of the schemas declared to define completely the XML files that support. Once the schema is defined, the associated files have no way to get out of this definition, so we can validate a XML given its schema. And this is not all. If you have an XML file, you can extract information and complete it to create a schema for that file.
Originally, with basic Petri nets, the structure of a Petri net was fully provided. The only thing that is not supported in comparison with graphic mode is the graphical appearance of Petri nets: the position of nodes and transitions was not important, but with the arrival of High level Petri nets and Petri nets design software, it is necessary to store this kind of information.
In this work, I am going to study only basic Petri nets, but that concept and the method is easily exportable to other kind of nets, such as Symmetric nets and High Level Petri nets (representable in PNML format too).

## 4. PNML

### 4.1. Description

Petri Net Marked Language is an xml language created to represent Petri Nets. With it, we can take a Petri Net and store it into an xml file without loss of information.
One of the best properties of PNML is that, as it is an xml based schema, it can be extended with more functionality extending the grammar. Virtually, any extension over Petri nets can be translated into PNML in a logical and natural way.
Moreover, this extension is defined by Petri net type definition (Billington et al 2003; Iso/iec 15909-2:2011}.
In this case, PNML hasn't got a way to represent subnets. There is something named `<page>` that is used to represent several nets in the same PNML file. But, by default, a node inside a page cannot connect with a node of other page. So it cannot be used as "subnets". So I will extend the language in order to get several goals:

1. Represent subnets of a Petri Net.
2. Include input and output interfaces for every subnet.

As we can think, definition of several subnets of a Petri net is possible and the connection over them is always through their respective interfaces.

### 4.2. PNML grammar

As PNML is an xml based language it has to be described by a schema that define the creation rules of the PNML representation of a Petri net.
The grammar is defined since 2009 and updated until 2012, which is the most recent revision. I am not going to do and extensive explanation of all the possibilities of the grammar, but the most important. As we can see later, anything we think is useful can be added to the process with little effort.
So I am going to study only the most basic elements of a Petri net. The rest of the element can be attached later with facility.

#### 4.2.1. PNML basics

In this section I am going to explain several characteristics of PNML files. With these explanations it is going to be easier the understanding of PNML structure.
First of all, as PNML files are xml files, there are several things to comply:

1. A xml file normally starts with a line defining some characteristics of the file, like the version and the encoding type. It has an aspect like this (Note: for clarity, in the following examples, this line can be deleted.):
   `<?xml version="1.0" encoding="utf-8"?>`
2. A root node must exist. In this case, the root node is `<pnml>`. So every PNML files has to start with the tag `<pnml>` and end with the tag `</pnml>`. Below this tag, there is a new tag `<net>` that can contain:
   (a) Type: the type of the Petri net as an attribute. In this case, as I am going to study only Place/Transition nets, it will be `ptnet`.
   (b) Name of the net: New tag `<name><text>...</text></name>`
   (c) Pages: one page is an invention to store several Petri nets inside an unique PNML file, but usually there is only one page for file. It is nested inside a `<page>` tag.

Furthermore, we cannot link elements from different pages.

3. Each element in PNML has to have a unique id inside the net to be identified unambiguously. So there cannot be two elements with the same id.

With these three observations, we can have an idea about how a PNML file is structured:

```
<?xml version="1.0" encoding="utf-8"?>
<pnml>
  <net id="myNet"
    type="http://www.pnml.org/version-
2009/grammar/ptnet">
    <name>
      <text> My new net </text>
    </name>
    <page id="page1">
      .......
    </page>
  </net>
</pnml>
```

Figure 7: Example of general PNML file

Once this structure is defined, I am going to explain the next stage, which is the most important one in this work.

#### 4.2.2. Places, transitions and arcs in PNML

As Petri nets have three main elements (places, transitions and arcs), PNML has them too. These three elements have several things in common in a PNML file:

- They are all nested in a tag `<page>`.
- Places and transitions (not arcs) can contain a tag `<name>` with its name. This tag has been defined before for the net's name. It can store information about the text of the name and the graphical position of this label in this way:

```
<name>
  <text> Element Name </text>
  <graphics>
    <offset x="22" y="-10"/>
  </graphics>
</name>
```

- They can contain information about its position and dimension in a tag `<graphics>`:

```
<graphics>
  <position x="100" y="200"/>
  <dimension x="40" y="40"/>
</graphics>
```

These are the common properties of places, transitions and arcs. Now let's go on the particular characteristics of each one of them.

Places are represented with the tag `<place>` and the can have a marking with the tag `<initialMarking>`.

Here we have an example of a place with two tokens in PNML:

```
<place id="p1">
  <name>
    <text> Place number one </text>
    <graphics>
      <offset x="130" y="130"/>
    </graphics>
  </name>
  <graphics>
    <position x="130" y="90"/>
    <dimension x="40" y="40"/>
  </graphics>
  <initialMarking>
    <text> 2 </text>
  </initialMarking>
</place>
```

Figure 8: PNML representation for places

Transitions are represented with the tag `<transition>`. Except the initial marking, it is really similar to a place. This is an example of a transition in PNML:

```
<transition id="t1">
  <name>
    <text> Transition number one </text>
    <graphics>
      <offset x="270" y="140"/>
    </graphics>
  </name>
  <graphics>
    <position x="270" y="100"/>
    <dimension x="40" y="40"/>
  </graphics>
</transition>
```

Figure 9: PNML representation for transitions

Arcs are represented with the tag `<arc>`. Arcs must have a source and a target, which are defined by the attributes `source` and `target` that have to point to a transition and a place, identified by their `id`. Furthermore, the arc weight can be fixed by the tag `<inscription>`. If the weight is one, the tag inscription is not necessary because this is the default value. This is an example of the arc with weight 3 that connects the place and the transition of the previous examples in PNML:

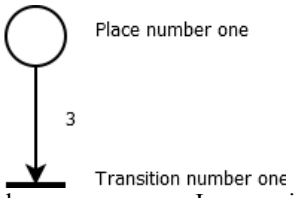
```
<arc id="a1" source="p1" target="t1">
  <inscription> 3 </inscription>
</arc>
```

Figure 10: General PNML representation for arcs with arbitrary weight

And this is the same example but with weight 1, that is obviated:

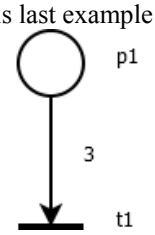
```
<arc id="a1" source="p1" target="t1"/>
```

If we take the last examples all together, we can represent the following Petri net:




For clarity and space reasons, I am going to obviate several options. I am not going to draw the graphic information and the names, but the id. And moreover,

the tags <?xml>, <pnml>, <net> and <page> are going to be obviated too. In many of the later examples they will not be present. So this last example is as follows:




```
<place id="p1">
  <initialMarking>
    <text> 2 </text>
 </initialMarking>
</place>
<transition id="t1"/>
<arc id="a1" source="p1" target="t1">
  <inscription> 3 </inscription>
</arc>
```

Figure 11: Simplified PNML representation for a basic Petri net

### 4.3. PNML extension for representing subnets

In this section I am going to define new tags and structures in PNML. At this point, I have developed all the necessary to extend PNML in order to represent subnets inside a concrete Petri net.

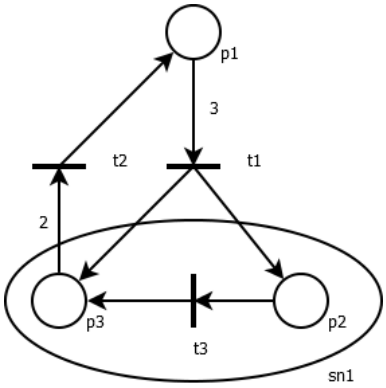



Figure 12: Subnet to represent in PNML

Let's take simple Petri net of the figure 12. It will serve us to explain the method to achieve a subnet representation and the PNML extension associated to it parting from a determinate Petri net:
The PNML code for this net is:

```
<place id="p1"/>
<place id="p2"/>
<place id="p3"/>
<transition id="t1"/>
<transition id="t2"/>
<transition id="t3"/>
<arc id="a1" source="p1" target="t1">
  <inscription>
    <text> 3 </text>
  </inscription>
</arc>
<arc id="a2" source="t1" target="p2"/>
<arc id="a3" source="t1" target="p3"/>
<arc id="a4" source="p3" target="t2">
  <inscription>
    <text>2</text>
  </inscription>
</arc>
<arc id="a5" source="t3" target="p3"/>
<arc id="a6" source="p2" target="t3"/>
<arc id="a7" source="t2" target="p1"/>
```

I want the ellipse region to be a subnet, so I have to specify a subnet with the elements inside the ellipse.
The first step is to define a new tag <subnet>. This tag will have an id, as the rest of PNML elements. And now we proceed in this way:

1. The places and transitions inside the subnet are moved into the tag <subnet>.
2. The arcs linking two elements that are both inside the subnet will be moved to into this new tag too.
3. The arcs entering or leaving the subnet will be copied inside the tag. This means that there are arcs duplicated inside and outside the tag.

If we apply these rules to the example:

1. p2, p3 and t3 are moved into the tag <subnet>.
2. a5 and a6 are put inside the tag.
3. a2, a3 and a4 are copied inside the tag.

And we have this other PNML extended code:

```
<subnet id="sn1">
  <place id="p2"/>
  <place id="p3"/>
  <transition id="t3"/>
  <arc id="a2" source="t1" target="p2"/>
  <arc id="a3" source="t1" target="p3"/>
  <arc id="a4" source="p3" target="t2">
    <inscription>
      <text> 2 </text>
    </inscription>
  </arc>
  <arc id="a5" source="t3" target="p3"/>
  <arc id="a6" source="p2" target="t3"/>
</subnet>
<place id="p1"/>
<transition id="t1"/>
<transition id="t2"/>
<arc id="a1" source="p1" target="t1">
  <inscription>
    <text> 3 </text>
  </inscription>
</arc>
<arc id="a2" source="t1" target="p2"/>
<arc id="a3" source="t1" target="p3"/>
<arc id="a4" source="p3" target="t2">
  <inscription>
    <text> 2 </text>
  </inscription>
</arc>
<arc id="a7" source="t2" target="p1"/>
```

Now I will separate the inside and the outside of the subnet completely. Taking advantage of the process described in section 2, I can extract its front-end.
In this case I have two igp (input gate to a place) and an ogt (output gate to a transition) with weight 2.

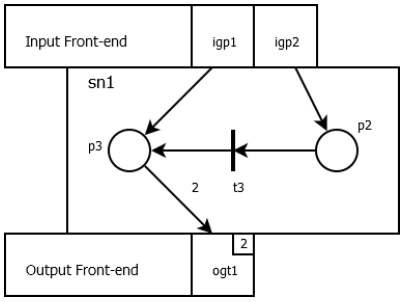



Figure 13: Subnet with its front-end

This is the complete net including subnet and front-end.

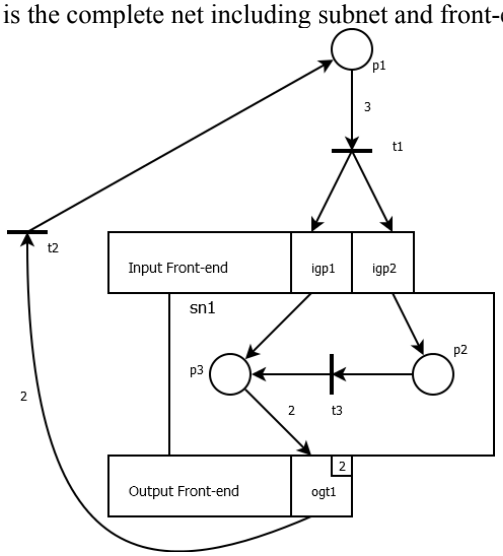



Figure 14: Petri net with subnet

And now that I have the graphic, how can I represent it in PNML? To answer this question I will define four new tags: `<interface>`, `<gate>`, `<inscription>` and `<content>`. Let's explain them.

As its name says, `<interface>` is the tag name for encapsulate the front-end. This tag has no attributes (just the id, of course) but it has embedded the gates inside of it. These gates are represented by `<gate>`. This tag has two new attributes: `action` and `type`. These two attributes have information about the gates. The attribute `action` can take two different values: `input` and `output`. It indicates whether the gate is an input or an output gate. The other attribute, `type`, can take other two values: `place` and `transition`.

As arcs have weight, gates have it too. For being in accordance, I define the tag `<inscription>` embedded in the tag `<gate>`. It has the weight of the arc associated. If the weight is 1 this tag can be obviated.

There is one other tag `<content>` that probably at this moment seems useless, but it is necessary for the rest of the process, as we will see in next sections. So I am going to introduce it now. This tag is used to encapsulate the rest of the subnet outside the interface. That is, `<subnet>` has two children: `<interface>` and `<content>`, that have the input/output place/transition gates and the rest of the elements, respectively.

At this moment I have to do only one thing more. The last step is to modify the arcs that are repeated inside and outside the net changing their id and source or target, depending on where is it:

- Change the id of one of the copies of the arc.
- If the arc is <u>entering</u> the subnet
  - For the `<arc>` tag inside the tag `<subnet>`, the `source` attribute of the arc is changed by the `input` gate associated
  - For the `<arc>` tag outside the tag `<subnet>`, the `target` attribute of the arc is changed by the `output` gate associated
- If the arc is <u>leaving</u> the subnet
  - For the `<arc>` tag inside the tag `<subnet>`, the `target` attribute of the arc is changed by the `input` gate associated
  - For the `<arc>` tag outside the tag `<subnet>`, the `source` attribute of the arc is changed by the `output` gate associated

Applying again all these rules to the example we have the definitive code for this Petri net:

```
<subnet id="sn1">
  <interface id="sn1-interface">
    <gate id="igp1"
         action="input" type="place"/>
    <gate id="igp2"
         action="input" type="place"/>
    <gate id="ogt1"
         action="output" type="transition">
      <inscription>
        <text> 2 </text>
      </inscription>
    </gate>
  </interface>
  <content id="sn1-content">
    <place id="p2"/>
    <place id="p3"/>
    <transition id="t3"/>
    <arc id="sn1-a2" source="igp2" target="p2"/>
    <arc id="sn1-a3" source="igp1" target="p3"/>
    <arc id="sn1-a4" source="p3" target="ogt1">
      <inscription>
        <text> 2 </text>
      </inscription>
    </arc>
    <arc id="a5" source="t3" target="p3"/>
    <arc id="a6" source="p2" target="t3"/>
  </content>
</subnet>
<place id="p1"/>
<transition id="t1"/>
<transition id="t2"/>
<arc id="a1" source="p1" target="t1">
  <inscription>
    <text> 3 </text>
  </inscription>
</arc>
<arc id="a2" source="t1" target="igp2"/>
<arc id="a3" source="t1" target="igp1"/>
<arc id="a4" source="ogt1" target="t2">
  <inscription>
    <text> 2 </text>
  </inscription>
</arc>
<arc id="a7" source="t2" target="p1"/>
```

Figure 15: Final PNML subnet representation

Once this is done, the only way to enter or leave the subnet is crossing the front-end.

This is a comprehensive definition of how to represent subnets in PNML. Now there are several ways to create a grammar extension that frame this structure of xml. For example, we can define a dtd file, a xsd file or, by coherence with the original grammar of PNML, a Relax NG file. I have defined a way to represent subnets, but the formal grammar is outside the scope of my work because of the wide casuistry of these Petri net types. However, the method is explained enough in order to each one of these types to define their own extension.

## 5. SECURITY

This is the second main goal, after subnetting. Once the possible subnets are defined it is the turn of securing them. It is possible to secure subnets or the entire net.

With secure, I mean four goals:

- Privacy. Concrete parts of the net must be occulted: the content is secret, so not everybody should be able to know it.
- Integrity. Any change in the secured parts has to be detected. If any of these parts suffers any kind of modification, the information may have been compromised, and perhaps it is not valid or correct. But I cannot know what has been modified: I can only detect that the original content has been changed.
- Authentication. I can authenticate the source of that net/subnet (signer, author or guarantor).
- Non repudiation. With this characteristic, the possibility of supplant other people is avoided. So the person that signs that part can't say that he hadn't done it. The signer cannot deny it.

But this information should be accessible to authorized people without necessity of supplying any other kind of data. So the whole information may be stored in the same file.
In the same way, there can be reasons for the rest of the security characteristics. For example, suppose that we have a Petri net that several people can access and:

- Some parts of that Petri net have been validated an accepted, so I want nobody to change them. In this case `integrity` is needed.
- I want to know who has developed a concrete chunk of the net. `Authentication` is required.
- There is a part of the Petri net that is bad defined and goes wrong. The person responsible of this part says that he hasn't made it and somebody has supplanted him. Then, `non-repudiation` is needed.

The best way to reach these goals is using standard and proved technologies. In this case, the selected technologies are:

- XMLEncryption (Xml encryption syntax and processing version 1.1, 2013) for privacy.
- XMLSignature (Xml signature syntax and processing version 1.1, 2013) for integrity, authentication and non- repudiation.

### 5.1. XMLEncryption

XMLEncryption is a World Wide Web Consortium (W3C) Recommendation for encrypting xml or non xml content. It is a standard xml file cipher. Both symmetric and asymmetric ciphering can be used. The main idea of this encryption is to replace the xml element or elements we want to be ciphered by other xml code that contains the ciphered data, in addition to information of the algorithms and keys used for the encryption process. When a non xml file is ciphered, the only option is to encrypt it completely. But, when it is applied to xml content, this technology allows us to define concrete fragments of the document we want to hide. Moreover, the xml document can be transformed before applying the encryption, for example, in order to normalize the xml content.
In this work, the pieces of xml content susceptible to be ciphered are, obviously, the subnets represented in PNML format.
Regardless of the data source (xml or non xml) the result is always a xml element. Normal is that this xml encrypted chunk has the whole necessary information to be decrypted. Among that information we can find:

- Ciphering algorithm: it is the name of chosen method to encrypt the data. It can be not included. In this case, both ciphering and deciphering agents have to know which is the exact this algorithm.
- The ciphered data: obviously this part is mandatory and has always to be present.
- Name of the chosen key: it is optional. It is used when a set of keys is known by both ciphering and deciphering agents.
- Key: it is optional. In this case there is a symmetric key in order to encrypt the data and an additional pair of keys: one (known by the cipher agent) to encrypt the symmetric key and the other (known by the decipher agent) to decrypt it.

Actually, there are several options to apply XMLEncryption, such as the algorithm or the key. The exact election of those option values is responsibility of the Petri net sender.
This section does not want to be an extensive explanation about XMLEncryption but a general idea about its functionality. So I am not going to deepen the whole characteristics of XMLEncryption. The final decision about which options use is responsibility of those people that want to apply this work, basing their decision on the requirements of their own Petri net.
Petri subnets are represented by a `<subnet>` tag that contains `<interface>` and `<content>`. This last tag contains the xml content that is going to be ciphered. Obviously, if we encrypt the interface we will have no way to connect the subnet with the rest of the net. So here is the utility of `<content>` tag.
Let's take the example of the figure 14 and its PNML representation. The goal is to hide the internal content of the subnet. If we apply XMLEncryption to the data contained inside the `<content>` tag, we will get something like this, depending on the algorithm and key selected for the ciphering:

```
<subnet id="sn1">
 <interface id="sn1-interface">
  <gate id="igp1" action="input" type="place"/>
  <gate id="igp2" action="input" type="place"/>
  <gate id="ogt1" action="output" type="transition">
   <inscription>
    <text> 2 </text>
   </inscription>
  </gate>
 </interface>
 <content id="sn1-content">
  <xenc:EncryptedData
```

```
    xmlns:xenc=http://www.w3.org/2001/04/xmlenc#
     Type="http://www.w3.org/2001/04/xmlenc#Element">
   <xenc:EncryptionMethod
     Algorithm=http://www.w3.org/2001/04/xmlenc#aes128"
     xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" />
   <xenc:CipherData
     xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <xenc:CipherValue
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
Wr1njyJlYYOM9lAYqcwGCWkw2L4pUjQD2GGVoU9lVZ0wKqHY8y3lGY8FY4i5K
3GY8FY4i5K3G8grIe1HRFqe7RtkFiXZgGMeYnQp6oB6ckKp3KFKHVqtucc9rA
VzOgC7XAwe61HRFqe6RRVzXjNM9hlVZ0wKqHY8y3l3GY8FY4i5K3G8grIe2xN
4u7x7fRtkFiXZgGMeYnQp6oB6ckKp3KFRRVzXjNAtVzOgC7XAw/oe61HRFqe6
RRVzXjNMLU5ZgGMeYny8NVPQmUSDX7NRtnR6YnQp6oB6GY8F=
    </xenc:CipherValue>
   </xenc:CipherData>
  </xenc:EncryptedData>
 </content>
</subnet>
<place id="p1"/>
<transition id="t1"/>
<transition id="t2"/>
<arc id="a1" source="p1" target="t1">
  <inscription>
    <text> 3 </text>
  </inscription>
</arc>
<arc id="a2" source="t1" target="igp2"/>
<arc id="a3" source="t1" target="igp1"/>
<arc id="a4" source="ogt1" target="t2">
  <inscription>
    <text> 2 </text>
  </inscription>
</arc>
<arc id="a7" source="t2" target="p1"/>
```

Figure 18: Ciphered Petri subnet content

The subnet content has vanished and replaced by xml code, corresponding to the ciphered data.

### 5.2. XMLSignature

Although privacy is a solved question with XMLEncryption, there are other aspects of the security that XMLEncryption can't cover that are: integrity, authentication and non-repudiation.

In this case I want to sign the whole Petri net or concrete parts of it. Obviously, if I sign only a fragment of a Petri net, this part keeps integrity, authentication and non-repudiation, but the rest of net doesn't.

As with XMLEncryption, the best way to achieve these goals is using standard technologies. For signing, the method chosen is XMLSignature.

With XMLSignature we can sign any kind of content but the result is XML content. It requires the use of digital certificates and a set of public/private keys, using asymmetrical ciphering algorithms for the process.

This signature can be enveloped, enveloping or detached, but they are very similar. I am going to use enveloped signature

A XML enveloped signature consists of `<Signature>` tag that is defined in the namespace: http://www.w3.org/2000/09/xmldsig\# and attached to the original xml file before closing the root node.

XMLSignature forces a digital signature to have:

- Canonicalization method: All the equivalent xml files are transformed in the same representative.
- Reference: Each reference indicates a part of the document that has to be signed. The set of all of these references are signed together and generates only one signature.
- Key information: Optionally, the signature can include necessary information to be validated, for example, with X.509 digital certificates.
- Transforms: It is a ordered list of processing steps that has to be applied to the content before being signed.

Many times we will need to sign the whole Petri net, but it in this case I want to sign certain parts of a Petri net, for example a critical subprocess. The modus operandi here is similar to XMLEncryption. First of all, the content to be signed should be grouped in a subnet and then, this subnet is signed.

The standard way to indicate a subnet to sign in XMLSignature is through a XPath expression. In XMLSignature, the way to specify XPath addresses is using XMLSignature XPath Filter . XPathFilter returns the node set that is going to be signed and it is placed into `/Signature/SignedInfo/Reference/Transforms` as a new `<Transform>`.

I am not going to explain all the possibilities of XPath Filter. I will explain only those main configurations useful to my objective.

The exact configuration depends on the particular necessities of each case.

The goal in this work is the signing of a concrete subnet. In this case, it is a little different as in XMLEncryption. Remember that in XMLEncryption, if I want to mask a subnet I don't process the `<subnet>` tag but the `subnet/content`. This is because the interface has to be visible. But in a signature I want to sign the complete subnet, including the interface. Suppose that this subnet has `id="sn1"`. The XPath expression that represents it is: `/pnml/net/page/subnet[@id="sn1"]`.

And the result of signing the subnet of the figure 14 is:

```
<?xml version="1.0" encoding="UTF-8"?>
<pnml>
 <net   id="myNet"   type="http://www.pnml.org/version-
2009/grammar/ptnet">
  <name>
   <text> My new net </text>
  </name>
  <page id="page1">
    .............
   <!—This is the same content as in the figure 15 -->
    .............
  </page>
 </net>
 <ds:Signature
     xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
   <ds:CanonicalizationMethod
      Algorithm="http://www.w3.org/TR/2001/REC-xml-
c14n-20010315"/>
   <ds:SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-
sha1"/>
   <ds:Reference URI="">
    <ds:Transforms>
     <ds:Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"/>
     <ds:Transform
Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315#WithComments"/>
     <ds:Transform
Algorithm="http://www.w3.org/2002/06/xmldsig-filter2">
```

```
      <dsig-xpath:XPath                    xmlns:dsig-
xpath="http://www.w3.org/2002/06/xmldsig-filter2"
Filter="intersect">
       /pnml/net/page/subnet[@id="sn1"]
      </dsig-xpath:XPath>
     </ds:Transform>
    </ds:Transforms>
    <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
    <ds:DigestValue>
     prCzhLgTCZ1ck6MjQnFy6cASCZw=
    </ds:DigestValue>
   </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>
QoO7mQmGBFTg2UxgiZnzlsnKi8V477JC0v12JPItL53zIOCpjhOwLoyxENl6v8lC
r3GdqrgZimNXMUjwR4zkd9FVNcIrn85DuRjHA/zDwSuPMq9w0N5A07c0xJ24uvn9
+zpbQxfblYTbkiy08+S0pqczU/bv5+g=
  </ds:SignatureValue>
  <ds:KeyInfo>
   <ds:X509Data>
    <ds:X509Certificate>
MIICgTCCAeqgAwIBAgIETfh4CTANBgkqhkiG9w0BAQUFADCBhDELMAkGA1UEBhMC
BAgTCExBIFJJT0pBMREwDwYDVQQHDAhMT0dST8KlTzEgMB4GA1UEChMXVU5JVkVS
TEEgUklPSkExDDAKBgNVBAsTA1BGQzEfMB0GA1UEAwwWScKlSUdPIExFw6BOIFNB
Fw0xMTA2MTUwOTE0NDlaFw0xMTA5MTMwOTE0NDlaMIGEMQswCQYDVQQGEwJFUzER
TEEgUklPSkExETAPBgNVBAcMCExPR1JPwqVPMSAwHgYDVQQKExdVTklWRVJTSURB
SU9KQTEMMAoGA1UECxMDUEZDMR8wHQYDVQQDDBZJwqVJR08gTEXDoE4gU0FNQU5J
CSqGSIb3DQEBAQUAA4GNADCBiQKBgQChePFNVCIfphFlyXQ9BysiR5BfXIuv3AnA
nVUjJeGnkUYQO32oUu+fEBK8WsEqjeH8A7zrHTRQjfYZWyuGWrM8gJXOa/P0MROP
1/+zLwR0tYkqLI2xqDOFII2RwK5L2yGeV4T4y8i3h1U0OFTSEwIDAQABMA0GCSqG
A4GBAIDOvAAdOCaTpy+83bGB2KmngMJrNxxWDpAi5LGFrN8iCShmbTpIeIbYBUAa
wD5QOENSFipQcdH5GEpPM9Rquy6xMwfda9EU5UfOSEmbk4fK2vaIOVjynpQsJ9P9
9enO2smQlyvw=
    </ds:X509Certificate>
   </ds:X509Data>
   <ds:KeyValue>
    <ds:RSAKeyValue>
     <ds:Modulus>
oXjxTVQiH6YRZcl0PQcrIkeQX1yLr9wJwCvNBbsOLUxcAp1VIyXhp5FGEDt9qFLv
/AO86x00UI32GVsrhlqzPICVzmvz9DETj5v3Px/G+Wujcdf/sy8EdLWJKiyNsagz
hSCNkcCuS9shnleE+MvIt4dVNDhU0hM=
      </ds:Modulus>
      <ds:Exponent>AQAB</ds:Exponent>
    </ds:RSAKeyValue>
   </ds:KeyValue>
  </ds:KeyInfo>
 </ds:Signature>
</pnml>
```

## 6. CONCLUSIONS

I have explained a comprehensive method to define subnets with interfaces. The interaction between these subnets and the rest of the net is always through this front-end. The selected representation method selected has been PNML. Once this is done, the conclusion of this work is to show that it is possible to apply security measures to Petri nets in order to fulfill four characteristics to Petri nets, or parts of it:

1. Privacy: not everybody can access some data
2. Integrity: not allowed changes are detected
3. Authentication: ensure who guarantee some information
4. Non-repudiation: one person cannot supplant other one

With XMLEncryption we achieve privacy. Integrity, authentication and non-repudiation are achieved with XMLSignature

## REFERENCES


Silva, M., 1985. Las Redes de Petri: en la Automática y en la Informática. Madrid: Editorial AC

León, I., 2011. Security and Protection in Petri Nets sending and storage. Signing and Encryption. Proceedings of EMSS, September 12-15, Rome (Italy)

León, I., 2013. Analysis of information partial encryption options for exchanging Petri Nets systems. Proceedings of EMSS , September 25-27, Athens (Greece)

León, I., 2014. Petri net representation with ciphered subnets: definition of PNML extensions for subnets representation and use of XMLEncryption for ciphering. Proceedings of EMSS, September 10-12, Bordeaux (France)

Valette, R., 1979. Analysis of petri nets by stepwise refinements. Journal of Computer and System Sciences, 18(1):35–46.

Suzuki, I and Murata, T., 1983. A method for stepwise refinement and abstraction of petri nets. Journal of Computer and System Sciences, 27(1):51–76.

Fahmy, H.M.A., 1990. Analysis of petri nets by partitioning: Splitting transitions. Theoretical Computer Science, 77(3):321–330.

Druzhinin, Va and Yuditskii, 1992. Construction of well-formed petri nets from standard subnets. Automation and Remote Control, 53(12):1922–1927.

Fahmy,HMA,1993.Analysi of petri nets by partitioning: splitting places or transitions. International Journal of Computer Mathematics, 48(3-4): 127–148.

Xia, C., 2011. Analysis and application of petri subnet reduction. Procs. of the IEEE, 6 (8):1662–1669.

Murata, T., 1989. Petri nets: properties, analysis and applications. Proceedings of the IEEE, 77(4):541–580.

Silva, M., 1993. In Practice of Petri Nets in Manufacturing. Chapman and Hall, London, UK.

David, R. and Alla, H., 2010.. Discrete, Continuous and Hybrid Petri Nets. Springer, Berlin, Germany, 1st ed., 2004 edition.

Peterson, JL., 1981. Petri Net Theory and the Modeling of Systems. Prentice Hall, Englewood Clifs, NJ.

Jensen, K. and Kristensen L.M., 2009. Coloured Petri Nets: Modelling and validation of concurrent systems.

Billington, J., Christensen,S., Van Hee, K., Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C. and Weber, M., 2003. The petri net markup language: Concepts, technology, and tools. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2679: 483–505.

Iso/iec 15909-2:2011 - systems and software engineering – high-level petri nets – part 2: Transfer format. http://www.iso.org/iso/catalogue detail.htm?csnumber=43538. [Online; accessed 21-May-2015].

Xml encryption syntax and processing version 1.1, 2013. URL http://www.w3.org/TR/xmlenc-core1/. [Online; accessed 20-Jun-2015].

Xml signature syntax and processing version 1.1, 2013. URL http://www.w3.org/TR/xmldsig-core1/. [Online; accessed 20-Jun-2015].