‘FAIR’ BPEL PROCESSES TRANSACTION USING NON-REPUDIATION PROTOCOLS

By

MUHAMMAD BILAL

Bachelor of Mechanical Engineering

University of Engineering & Technology

Lahore, Pakistan

2000

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May 2005

'FAIR' BPEL PROCESSES TRANSACTION USING NON-REPUDIATION PROTOCOLS

Thesis Approved:

Johnson Thomas

Thesis Adviser

John P. Chandler

Debao Chen

A. Gordon Emslie

Dean of the Graduate College

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Development of web services is a step toward universal interoperability between applications by using web standards. The complete prospective of web services as an integration platform can only be accomplished by combining the complex interactions among the applications and business processes using a standard process integrated model. These interactions typically suppose to be the sequences of peer to peer synchronous or asynchronous message exchanges. A business process requires a proper description of the message exchange protocol for interactions. In these protocols the message exchange behavior among parties should be visible without disclosing their internal implementation.

"*Business Process Execution Language for Web Services* usually called BPEL is a language for the formal specification of business processes and business interaction protocols" [2]. It expands the web services so that they can interact with each other and capable for supporting exchange of data, business documents, agreements, payments, contracts, acknowledgements, and business transactions among themselves. BPEL is an XML-based flow language. It defines the interaction of business processes. Complex processes can be created in BPEL by developing and integrating different activities e.g. different Web services invocations, manipulate data, throw faults, or end a process and these activities are nested within structured activities in order to run them parallel, or sequential or depending on required conditions.

“The role of BPEL4WS is to define a new Web service by composing a set of existing services to enable business processes to interoperate within and between companies that use different underlying technologies (universal interoperability)”[2].

Web services broadcast themselves so that other web services can discover them and also invoke them creating a communication amongst each other. BPEL provides an environment to describe business processes that include multiple Web services and standardize message exchange internally and among partners. Linking those web services together into a one large business process gave the user a number of disputing dilemmas.

One of these problems is Non-repudiation which means denial of having participated in a message exchange. Non-repudiation is one of the important security issues. “Non-repudiation is a security service which creates, collects, validates and maintain cryptographic evidence of an electronic transaction to support the settlement of a possible disputes “[1]. A numbers of protocols have been developed to solve Non-repudiation. In general, the messages are encrypted with a secret key and send it to the receiver.

Fairness of a protocol depends on who is controlling the execution of the protocol. It may be inclined either toward the sender or receiver, or may be fair to both. For example receiver repudiation can be avoided by designing a protocol such that the sender sends the encrypted message and does not release the encryption key until he gets a receipt acknowledgment from the receiver. Such a protocol is in the favor of the sender because he may not send key after receiving acknowledgement and claims that he did. On the other hand a Trusted Third Party (TTP) is involved which releases the key to the receiver and message no longer under control of sender after sending the encrypted message to the receiver which makes protocol fair for the receiver. To eliminate the presence of TTP at the time of dispute, the protocol needs to generate enough digital evidences for both the sender and the receiver.

Petri Net is a graphical and mathematical tool [10] used for modeling and analyzing system with concurrency. Petri Nets have been successfully used for modeling of communication protocols. Petri net consists of Places, Transitions and Arcs. Transitions are active components and on firing change the state of the system. The current state of the system is represented by number of tokens at a Place. Both Places and Transitions are connected by Arcs. Arcs from Places to Transitions are inputs and Arcs from transitions to Places are called outputs.

Motivation:

WS-security is used to make secure communication among BPEL processes but it does not provide fairness and accountability. To fulfill such requirements there is a need to use a fair non-repudiation protocol. In this thesis, we propose a number of Non-repudiation protocols for different scenarios. The proposed protocols are specified in BPEL because they provide security, accountability, fairness, timeliness and confidentiality. Furthermore we use Petri Net in order to analyze the proposed non-repudiation protocols that are specified in BPEL are correct.

The rest of the thesis is organized as follows: chapter 2 introduces the BPEL and basic notations used in web services and business processes. It also gives a brief introduction to Non-repudiation protocols. We describe the objective of this thesis in chapter 3. In Chapter 4 we propose the non-repudiation protocols, specify them in BPEL and analyze those protocols.

Chapter 2

Literature review

2.1 Literature review on BPEL4WS:

The BPEL is an XML-based standard that is used to combine Web services in order to develop business processes. “It provides an XML-based grammar for describing the control logic required to coordinate web services participating in a process flow and is layered on the top of WSDL, with BPEL defining how the WSDL operation should be sequenced” [2]. BPEL consist of two types of processes: executable and abstract.

Abstract process:

An abstract process is a business protocol, specifying the message exchange behavior between different parties without revealing internal activities. In BPEL process all partners and their interactions are represented in terms of abstract WSDL interface (portType and operations) with no reference to the actual services invoked by the process instance.

Executable process:

It specifies the execution order between a number of activities constituting the process, the partners involved in the process, the messages exchanged between these partners, and the fault and exception handling specifying the behavior in cases of errors and exceptions.

2.1.1 Relation with WSDL:

The interaction between the service and its partners are represented in the WSDL file. The portTypes, which reside in the WSDL file, are used as reference to define a BPEL process. Since the possible deployment of the process is not used to define BPEL process, it allows the reuse of business process definitions for several deployments.

2.1.2 Relation with SOAP:

There are operations in WSDL file that bound to a number of protocols, one of which is SOAP. BPEL does not specify which binding to use. There are ports in WSDL file and each port must be associated with a binding.

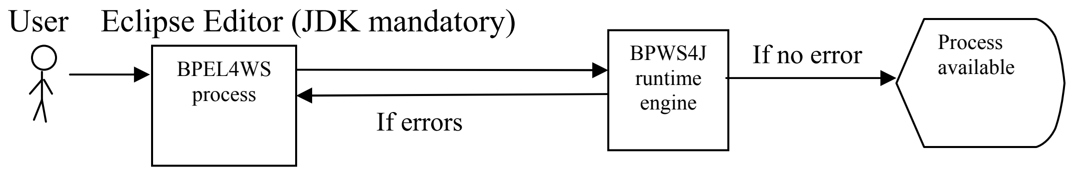
2.1.3 Security in BPEL:

"It is strongly recommended that business process implementations use WS-Security to ensure messages have not been modified or forged while in transit or while residing at destinations" [2]. WS-security provides security by secure SOAP message exchange.

2.1.4 BPWS4J: A runtime engine for BPEL

The BPWS4J is a runtime engine [27] that takes BPEL document which includes the process to be executed, the WSDL document of the services that process may invoke during execution, and the WSDL document that contain the interface information that the process will present to the client process. This runtime engine automatically generates the bindings that are required to interact and the process is available as a Web service with a SOAP interface. Runtime engine supports the invocation of Web services with SOAP interface, or EJBs, or normal Java classes.

BPWS4J is used for creating and executing BPEL4WS processes. Platform required for BPWS4J are Linux, Windows® 2000 or other platforms for which Eclipse 2.1 is available. Eclipse editor is used to create and edit BPEL files. JDK 1.3 is mandatory for Eclipse editor.



2.2 Literature review on non-repudiation protocol:

There are two approaches for fair non-repudiation. In general these protocols encrypt the message with a secret key and send it to the receiver and then two the parties exchange a delivery receipt and the message key to get the original message. An alternate approach is to involve a trusted third party that acts as a notary.

We consider different cases in message passing protocols [1, 3, 4, 15]. Assume that A wishes to send a message M to B and get the corresponding receipt. TTP is a trusted third party.

Basic Notations

eK(X) and dK(X): encryption and decryption of message X with key K.

sK(X): digital signature of message X with the private key K.

P_A, S_A: the public and private key of A.

s: signature

A → B: X : A send message X to B.

A ↔ B: X : A fetches message X from B.

NRO = sS_A (f_{NRO}, B, L, C): Non-repudiation of Origin of M.

NRR = sS_B (f_{NRR}, A, L, C): Non-repudiation of Receipt of M.

*sub*_K = sS_A (f_{SUB}, B, L, K): proof of submission of K.

*con*_K = sS_T (f_{CON}, A, B, L, K): confirmation of K issued by TTP.

f_{NRO}: flag information indicating NRO (Non-repudiation of Origin)

f_{NRR}: flag information indicating NRR (Non-repudiation of Receipt)

f_{SUB}: flag information indicating submission of key.

f_{CON}: flag information indicating confirmation of key issued by TTP.

f_{ACK}: flag information indicating acknowledgement.

Case 1: Sender and receiver are fair to each other and communicating channel is completely reliable. In this case, the protocol is very simple.

1. A $\longrightarrow$ B : f_{NRO}, B, M, sS_A (f_{NRO}, B, M)

Sender A is sending a message M signed with its private key to B. Here sS_A (f_{NRO}, B, M) is a digital signature on message (f_{NRO}, B, M) with the private key of A.

2. B $\longrightarrow$ A : f_{NRR}, A, sS_B (f_{NRR}, A, M)

Where, sS_B (f_{NRR}, A, M) is a digital signature on message (f_{NRR}, A, M) with the private key of B. The signatures of sender and receiver serve as evidence. Note that in the scheme the message is not encrypted.

Case 2: Sender and receiver don't necessarily play a fair role, but the communicating channel is completely fair. The above protocol is advantageous to B in a situation where A sends a message and B can refuse to send an acknowledgement. This problem can be solved by involving a TTP.

1. A$\longrightarrow$ TTP : f_{NRO}, TTP, B, M, sS_A (f_{NRO}, TTP, B, M)

2. TTP $\longrightarrow$ B : f_{NRS}, A, B, M, sS_T (f_{NRS}, A, B, M)

Where sS_T (f_{NRS}, A, B, M) is a digital signature of TTP using its private key.

3. TTP $\rightarrow$ A : f_{NRD}, A, B, sS_T (f_{NRD}, A, B, M)

Signatures of the TTP serve as proof of submission. Still this protocol is advantageous to B in a situation where B could deny having received the TTP's message.

Case 3: Sender and receiver are fair to each other but the communicating channel is not completely reliable. Each message can be sent over and over again until acknowledgment has been received.

1. A $\rightarrow$ B : f_{NRO}, B, M, sS_A (f_{NRO}, B, M)

2. B $\rightarrow$ A : f_{NRR}, A, sS_B (f_{NRR}, A, M)

B confirms receipt of A's message.

3. A $\rightarrow$ B : f_{ACK}, B, sS_A (f_{ACK}, B, M)

As the communication channel is not reliable, A informs B that it has received B's receipt of A's message. We assume that B repeatedly does step 2 until A's acknowledgment. In this protocol B has an advantage in a situation where B need not provide a proof of receipt.

Case 4: Both parties do not necessarily play fair and the communication channel is not reliable. To solve this problem, the encrypted message is sent under the key *K* that is sent later.

1. A $\rightarrow$ B : f_{POE}, B, eK (M), sS_A (f_{POE}, B, eK(M))

2. B $\rightarrow$ A : f_{ACP}, A, sS_B (f_{ACP}, A, eK(M))

3. A $\rightarrow$ B : f_{NRO}, B, K, sS_A (f_{NRO}, B, K)

4. B $\rightarrow$ A : f_{NRR}, A, sS_B (f_{NRR}, A, K)

Unfortunately, the problem is still there and B can refuse to send the last message, leaving A without proof of receipt.

2.3 J. Zhou and D. Gollmann's Protocol:

This protocol considers that the communication channel is not reliable and the two parties do not necessarily play fair. "A non-repudiation protocol is fair if it provides the originator and the recipient with valid irrefutable evidence after completion of the protocol, without giving any party an advantage over the other at any stage of protocol"[3].

2.3.1 ZG's fair protocol

In this protocol [3], the originator divides a message into two parts, a commitment C and a key K. C is a cipher text of message M, e.g. M encrypted with a key K. L is a unique label for the protocol run. This protocol assume that A, B and TTP are each equipped with their own private signature key and the related public verification keys and even in case of network failure, both parties will be able to retrieve the key from TTP. The main idea of this protocol is to send C first and then key K, which unlocks the message, is released:

1. A ⟶ B : f_{NRO}, B, L, C, NRO

2. B ⟶ A : f_{NRR}, A, L, NRR

3. A ⟶ TTP : f_{SUB}, B, L, K, *sub*_K

4. B ↔ TTP : f_{CON}, A, B, L, K, *con*_K

5. A ↔ TTP : f_{CON}, A, B, L, K, *con*_K

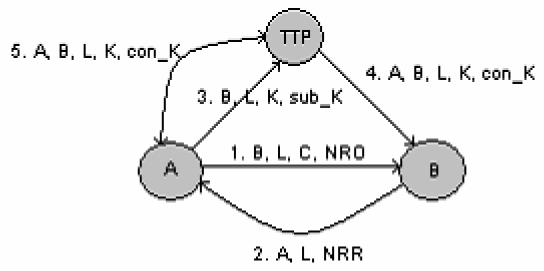


Figure 2.1: A Fair Non-repudiation Protocol

In step 1, A sends the encrypted message to B. In step 2, B confirms receipt, but cannot access the original message. In step 3, A submits the key to the TTP; *sub*_K is the proof of submission of K. The TTP stores the tuple (A, B, L, K, con_K) in some read only directory available to public; *con_K* is the confirmation of key issued by the TTP. In step 4, B gets the key and in step5 A confirms that B can indeed get the key.

The protocol is called "fair" but as pointed out in [4], the protocol has some weaknesses. First, the successful execution of the protocol depends on whether the sender submits the key to TTP. So it is advantageous to senders and the protocol is not fair to the message recipients. Secondly, the encrypted key K is visible to TTP, thus, anyone who can access the key at TTP is able to access the original contents of message M.

2.4 Non-repudiation Message Protocol for Collaborative E-business:

In this protocol [15], the message is encrypted with secret key, which is generated at runtime. The sender sends a message encrypted with the secret key. That key is 'double-encrypted' which means a twice-encrypted secret key that is first encrypted with the receiver's public key and then with the public key of TTP.

$$\text{Encrypted message} = \text{em} = \text{eK(M)}$$

$$\text{Double-encrypted Key} = \text{dek} = eP_{TTP}(eP_R(K))$$

Where M is the original message, K is secret key, P_R is private key of recipient and P_{TTP} is private key of TTP. The receiver can access original message only using secret key.

Step1: The sender generates the secret key randomly to encrypt the message. It then double-encrypt the secret key. Signature is generated on concatenation of the message digest of the encrypted message and the message digest of the double encrypted secret key. All this information is sent to the recipient.

$$S \longrightarrow R : t_{id} \mid S \mid em \mid dek \mid dual\_signature$$

Where [15], t_{id} = transaction id

$dual\_signature = t_{id} \mid md1 \mid md2 \mid sS_S(t_{id} \mid md1 \mid md2)$

md1= MD(em): message digest of the message

md2= MD(dek): message digest of the double encrypted key

Step 2: The recipient forwards double-encrypted key to the TTP along with its signature that he received a correct encrypted message as an acknowledgement. The recipient is required to send his signature, otherwise TTP will not reply to it.

$$R \longrightarrow TTP : t_{id} \mid S \mid R \mid md1 \mid dek \mid sS_R(t_{id} \mid md1)$$

Step 3: TTP decrypts the double-encrypted key and release the encrypted key to the recipient and wait for the acknowledgement from R for sometime.

$$TTP \longrightarrow R : t_{id} \mid ek\_from\_TTP$$

Where $ek\_from\_TTP = dS_{TTP}(\ dek\ )$: decryption of dek using private key of TTP.

Step 4: After getting the secret key and the message by decrypting it with its own private key, the receiver creates a signature on the digested secret key and sends it to TTP which is the conformation of receiving the secret key.

$R \longrightarrow TTP : t_{id} \mid sS_R(MD(ek\_from\_TTP))$

Step 5: In the final step of the protocol, TTP forwards two signatures that he has received in step 2 and 4 from recipient to the original sender.

$TTP \longrightarrow S : t_{id} \mid sS_R(t_{id} \mid md1) \mid sS_R(MD(ek\_from\_TTP))$

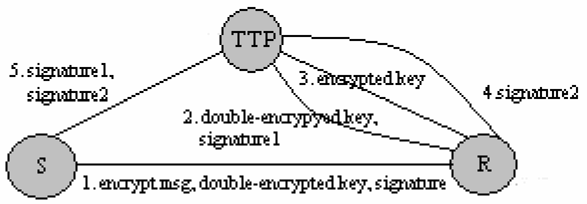

TTP
S
R
5. signature1, signature2
3. encrypted key
4 signature2
2. double-encrypted key, signature1
1. encrypt msg, double-encrypted key, signature

Figure 2.2: Secure message Transfer Protocol for E-commerce

Chapter 3

Objective

BPEL4WS is a model and a grammar for describing the behavior of a business process on interactions between the process and its partners and it involves invocation of different web services to achieve business goals. It interacts with partners and web services by message passing. There are two things to be considered.

- Usually messages at the business level may contain confidential types of documents so it is required that no one else can get the original messages except the partner to which it is sent.
- There is a problem of repudiation among parties.

To avoid these circumstances we propose a protocol that protects the confidentiality of message contents such that no unauthorized intermediary is able to see the original message and achieve non-repudiation by involving a trusted third authority (TTP) but do not need the third party at the time of dispute and more over the third party cannot access the secret key and hence the original message. This is done by encrypting the message with a secret key and then double encrypts that key. It means a twice-encrypted secret key that is encrypted with the recipient's public key first and then with the public key of Trusted third party (TTP) involved in the protocol.

We propose a novel Non-repudiation protocol for a chain linked business transactions that may involve more intermediate parties in many different topologies. In our protocol, intermediate parties may or may not be able to access and modify the original message depending upon the authorization needed. We extend this protocol to facilitate the sender to send different messages to the multiple entities. When there are multiple entities involved, there is a high probability of communication bottleneck created at the TTP. The proposed protocol aims to reduce this bottleneck. In the proposed protocol, the originator can send the messages to multiple recipients using a single key. The third party therefore needs to decrypt only a single key and not multiple keys from each entity involved in the business communication. We also propose a protocol with an intermediate entity which acts as a hub, thus reducing the load on the originator. In our chain- linked protocols, the recipient can get the key only after identifying himself, which means the key cannot be revealed to a wrong unidentified party. We analyze that protocol to make it optimal, improve, efficient, secure and accountable. We specify these non-repudiation protocols in BPEL.

This thesis also proposes a Petri Net merging method to model the integration of web services and business processes. We use Petri net to get the reliability of chain-linked business transaction. We use color Petri net to explain the flow of chain-linked protocol in which intermediate node can modify messages.

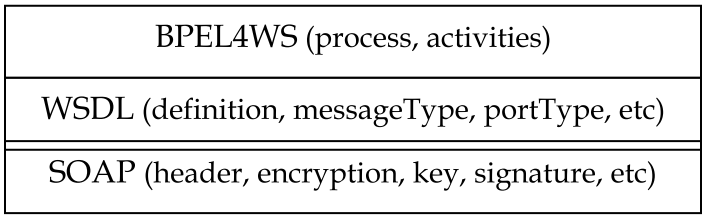
Chapter 4

Methodology and Approach

4.1 Assumptions:

1. Web services within the organization can trust each other. A non-repudiation protocol is therefore required only when communication is in between external services.
2. Third party is not available at the time of dispute.
3. Communication channels are reliable.
4. Key generating algorithms are out of scope.

BPEL is a layer on top of WSDL, that is, it uses WSDL to specify actions that should take place in a business process, and to describe the web services provided by a business process. There are ports in WSDL that must be associated with bindings, one of which is SOAP.


BPEL4WS (process, activities)
WSDL (definition, messageType, portType, etc)
SOAP (header, encryption, key, signature, etc)

4.2 Building reliable asynchronous process:

By default, web services perform synchronous operations. In our case we focus on security, fairness and accountability. Each process may be asynchronous, that is, the originator may start a business transaction with a receiver at any time. However, the non-repudiation protocol for each individual business transaction is synchronous. To enable asynchronous communication using web services, every party involved in the protocol should be implemented as a web services. This make a peers-to-peer process where all parties can initiate conversation when necessary.

4.2.1 Flow of a protocol

Brief description:

We consider a transaction between two parties, one is a Buyer and other is a supplier. We involve a Trusted Third Party to establish Non-repudiation between parties. We divide the Buyer BPEL process into two components because the requested business processes may take time for processing request. There are four public web services (considering BPEL process as a web service) and some internal web service. An internal web service, which is an inventory manager, sends an order to replenish inventory to the buyer request process. The buyer request process now needs to interact with the external web service. These are the steps executed between a requester and a supplier.

1. Requester sends a purchase message, which is encrypted with double-encrypted key (Key is first encrypted with the public key of the recipient and then with the public key of TTP) to the seller along with the double-encrypted key and dual signature (signature on the message digest of the double encrypted key and message digest of the encrypted message – see chapter 2).

2. Supplier receives the encrypted message and sends an acknowledgement receipt back to the requester after checking the integrity of main content eK(M) and double-encrypted key by comparing with the dual signature. Both eK(M) and the double-encrypted key are checked by the supplier generating the message digests and comparing with the message digests in the signature. So supplier confirms that it receives the correct encrypted message contents before proceeding.
3. Supplier forwards the double-encrypted key to the TTP, along with its signature1 on the message digest of encrypted message to acknowledge the correct receipt of the encrypted message. Supplier is required to send this signature1 to the TTP in order to access the key. TTP stores the signature1 temporarily for signature distribution at the end of the protocol.
4. TTP decrypt the double-encrypted key using its private key and releases the encrypted key to the Supplier. The TTP then waits for acknowledgement from the supplier. In case the TTP does not receive this acknowledgement within a certain timeout, TTP detects the supplier's misbehavior.
5. The supplier decrypts the encrypted key received form the TTP using its private key. It then sends signature2 on the message digest of the decrypted secret key to the TTP, as confirmation of receiving the key. The supplier creates the signature2 on the digested secret key so that TTP cannot access any key information from the signature.
6. TTP sends two signatures in step 2 and step 4 to the original sender. These two signatures are the supplier's acknowledgement of receiving correct purchase message and secret key.

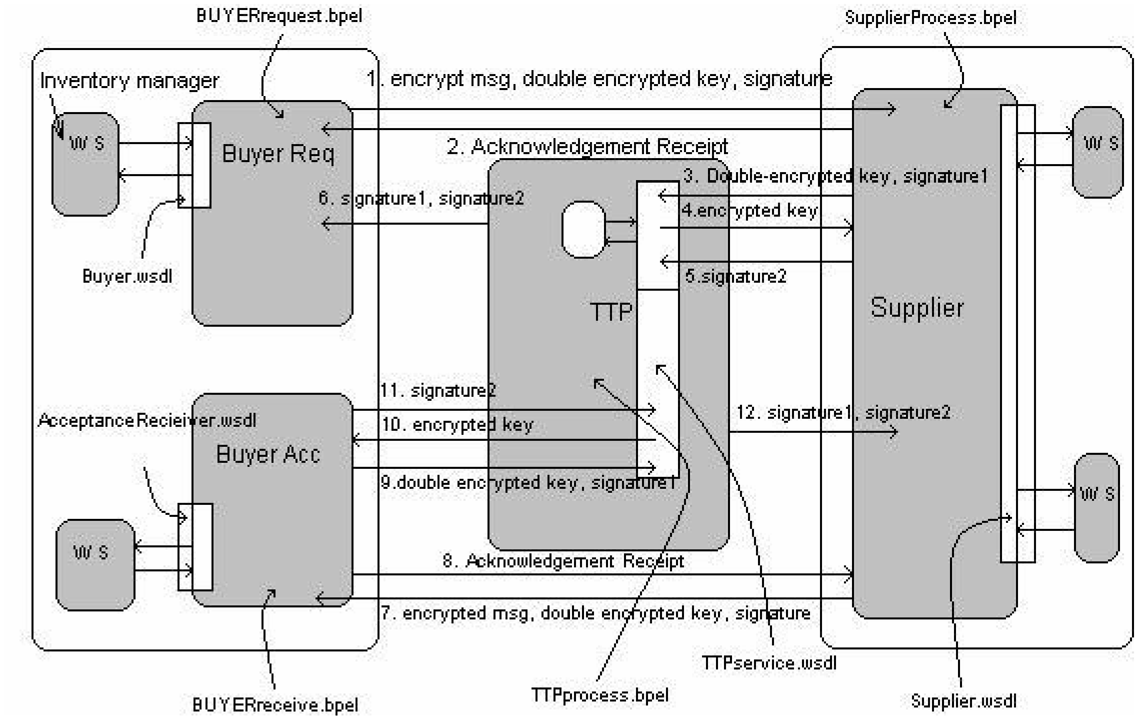

BUYERrequest.bpel
SupplierProcess.bpel
Inventory manager
W S
Buyer Req
1. encrypt msg, double encrypted key, signature
2. Acknowledgement Receipt
3. Double-encrypted key, signature1
4.encrypted key
5.signature2
6. signature1, signature2
Buyer.wsdl
TTP
Supplier
11. signature2
10. encrypted key
9.double encrypted key, signature1
12. signature1, signature2
AcceptanceRecieiver.wsdl
Buyer Acc
8. Acknowledgement Receipt
7. encrypted msg, double encrypted key, signature
TTPservice.wsdl
TTPprocess.bpel
BUYERreceive.bpel
Supplier.wsdl

Figure 4.1: The e-business dialogue

7. After processing the buyer's request, supplier sends the encrypted purchase acceptance message, along with double encrypted key and dual signature to second component of buyer as in step 1.
8. Buyer Acceptance process sends an acknowledgement receipt back to supplier.
9. This is same as step 3, but instead of supplier, this message is send by Buyer Receiver component to TTP.
10. TTP decrypt the double-encrypted key and release the encrypted key to the Buyer Receiver component process.
11. The Buyer Receiver component sends signature to the TTP, the confirmation of receiving the key same as in step 5.
12. The protocol ends with TTP forwarding both signatures to seller.

There are 12 messages. We can reduce it to 10 by removing step 2 and 8 and use step 6 and 12 as acknowledgments.

4.3 Building the Non-repudiation BPEL4WS Processes:

Conceptually, we need only three processes, one for buyer, one for supplier and one for TTP. The supplier may take time to process order so we need to split the buyer process as two separate BPEL4WS processes. We develop a business process to show a quick overview of the activities and syntax of BPEL.

4.3.1 Building the public Web service definitions:

Start from the WSDL definitions for processes, starting with Buyer.wsdl shown in figure 4.2. This web service allows replenishing inventory by placing order according to their need.

“BPEL4WS enabled WSDL definitions require some additional information and since this WSDL file will be used in the BPEL4WS process, it requires a service link definition to be added to the file. Service links enable partners in the BPEL4WS process to be linked to actual "actions" defined in the BPEL process.”[5].

Following is the Buyer.wsdl file. Logic is the same as in [5]

```
<definitions targetNamespace="http://www.buyer.com/services/Buyer"
          xmlns:BUYER="http://www.buyer.com/services/Buyer"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
          xmlns:slt="http://schemas.xmlsoap.org/ws/2002/07/service-link/"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

  <message name="ReplenishInventroyRequest">
    <part name="PORequest" type="xsd:string"/>
  </message>
  <message name="ReceiptAcknowledgement">
    <part name="receiptAck" type="xsd:string"/>
  </message>
<message name="SignaturefromTTP">
     <part name="SupSignatureOnPO" type="xsd:string"/>
  </message>

  <portType name="ReplenishInventoryRequestPort">
     <operation name="replenishRequest">
        <input message="BUYER:ReplenishInventoryRequest"/>
        <output message="BUYER:ReceiptAcknowledgement"/>
     </operation>
  </portType>
<portType name="SignaturefromTTPPort">
     <operation name="SignatureReceive">
        <input message="BUYER:SignaturefromTTP"/>
     </operation>
  </portType>

  <slt:serviceLinkType name="ReplenishInventoryRequestSLT">
    <slt:role name="inventoryService">
      <slt:portType name="BUYER:ReplenishInventoryRequestPort"/>
    </slt:role>
  </slt:serviceLinkType>

<slt:serviceLinkType name="SignatureTransmitReqSLT">
    <slt:role name="GetSignatureFromTTP">
      <slt:portType name="BUYER:SignaturefromTTPPort"/>
    </slt:role>
  </slt:serviceLinkType>
  <!—BPWS4J Engine will automatically generate the bindings-->
<service name="BUYERPORequesterServiceBP">
  </service>
</definitions>
```

Figure 4.2: Buyer.wsdl

There are three messages, a request for purchase order, acknowledgment receipt of the request, and signature from TTP.

Service links are used to define the capabilities of partners in the BPEL process. A partner is linked to a portType and also a set of operations in the WSDL file using those service links. In this example, Web service is an associated component of a BPEL process, the BPWS4J engine automatically generates the necessary bindings which helps the BPEL process to absorb the Web service, listening to the defined ports for any activity [5]. Therefore, the service and bindings sections of the WSDL definition are empty.

4.3.2 Request Component of Buyer:

This process takes the request from an inventory manager and sends request with double-encrypted key to the supplier. In the final step it receives signatures from TTP. There are three partners of the buyer request process - inventory manager, Supplier and TTP. We first define partners and containers to store data.

The BPEL process at the buyer is BUYERrequest.bpel. It is layered on top of the BUYER.wsdl file that we created earlier.

BUYERrequest.bpel is as follow. Logic is the same as in [5]

```
<process name="BUYERrequestProcess"
        targetNamespace="http://www.buyer.com/services/BUYERrequestProcess"
        xmlns="http://schemas.xmlsoap.org/ws/2002/07/business-process/"
        xmlns:buy="http://www.buyer.com/services/Buyer"
        xmlns:supply="http://www.supplier.com/wsdl/POService">
        xmlns:TTP="http://www.ttp.com/services/TTPservice">

<!-- This process has three partners, the internal inventory service that triggers the -->
<!-- process and the Supplier, to whom the PORequest is sent and TTP             -->

  <partners>
    <partner name="inventoryService"
                         serviceLinkType="buy:ReplenishInventoryRequestSLT"/>
    <partner name="supplier" serviceLinkType="supply:PORequestSLT"/>
    <partner name="TTP" serviceLinkType="TTP:TTPrequestSLT"/>
  </partners>

<containers>
  <container name="replenishRequestCTR"
                          messageType="buy:ReplenishInventroyRequest"/>
    <container name="replenishResponseCTR"
                          messageType="buy:ReceiptAcknowledgement"/>
    <container name="outputPORequestAckContainer"
```

```
                                messageType="supply:ReceiptAckPORequestType"/>
    <container name="SupplierSignaturefromTTP"
                                messageType="buy:SignaturefromTTP"/>
    </containers>

<sequence name="placePOSequence">

<!-- Receive the inventory manager PO Request and store it in replenishRequest
container-->
 <receive name="ReplenishRecieve"
           partner="inventoryService"    portType="buy:ReplenishInventoryRequestPort"
           operation="replenishRequest"
           container="replenishRequestCTR" createInstance="yes">
   </receive>

<!-- Initialize container -->
<assign>
      <copy>
        <from expression="'initializing'"/>
        <to container="replenishResponseCTR" part="receiptAck"/>
      </copy>
</assign>

<!-- Invoke placePORequest service at supplier Inc -->
<invoke name="PlacePOwithSeller"
          partner="supplier" portType="supply:placePORequestPort"
operation="placePORequest"
          inputContainer="replenishRequestCTR"
          outputContainer="outputPORequestAckContainer"/>

<!-- Copy acknowledgement receipt from supplier to be sent to Inventory Manager -->
<assign>
      <copy>
        <from container="outputPORequestAckContainer" part="receipt"/>
        <to container="replenishResponseCTR" part="receiptAck"/>
      </copy>
</assign>

<!-- Reply to inventory manager with the acknowledgement receipt of the PO Order
sent by supplier -->
<reply name="ReplenishResponse"
         partner="inventoryService"
         portType="buy:ReplenishInventoryRequestPort"
         operation="replenishRequest"
         container="replenishResponseCTR">
</reply>
```

```
<--    Process receive Signature from TTP      -->
<receive name="SignatureRecieve"
           partner="TTP" portType="buy:SignaturefromTTPPort "
           operation="SignatureReceive"
           container="SupplierSignaturefromTTP" createInstance="no">
  </receive>
</sequence>
</process>
```

Figure 4.3: BUYERrequest.bpel

In this process, after defining the partners and containers of the process, we specify the BPEL activities of the process start with the sequence activity.

Begin sequence

- Receive a request from the Inventory Manager and deposit it in the request container.
- Assign the data from the request container to the container to be sent to supplier.
- Invoke a "place purchase order" request with supplier based on data stored in the pervious step.
- Assign the acknowledgement received from supplier to container being sent to Inventory Manager.
- Reply to the Inventory Manager with the response from supplier.
- Receive the signatures from trusted third party show that supplier has accessed the original message.

End sequence

Note: Receive, Invoke, Reply, and Assign etc. are the BPEL activities. See appendix for more detail.

4.3.3 Supplier Process:

The sequence of activities of BPEL process at supplier is as follow:

Begin sequence

- Receive the Request from Buyer Request process.
- Assign a receipt message in the container that is used in the reply activity.
- Reply to Buyer Request process.
- Assign encrypted message and double encrypted key in the container that used in the invoke activity.
- Invoke the TTP process and send the Double-encrypted key and signature1.
- After receiving the key from the TTP it again invoke the TTP process to send the signature2.
- After processing the order send the order status to the Buyer Acceptance process by invoke.
- Finally, it receives the signatures of buyer receive process from TTP.

End sequence

4.3.4 Acceptance Component of Buyer:

The second BPEL process at BUYER has following sequence.

Begin sequence

- Receive the Acceptance from Supplier.
- Reply to the supplier with the receipt acknowledgement.

- Invoke the TTP to decrypt the double encryption key and send the signature1 (on the message digest of original message and dek), and to get the encrypted key.
- Invoke the TTP to send the signature2 (on the digested key) after accessing original message.

End sequence

4.3.5 TTP Process:

The Fourth BPEL process places at the TTP.

Begin sequence

- Receive the Request and signature1 from the supplier process to decrypt the key.
- Reply to the request from the supplier process.
- Receive the signature2 from the supplier process.
- Invoke the Buyer Request process to send the signatures of the supplier process.
- Receive the request and signature1 from Buyer Acceptance process to decrypt the key.
- Reply to the request from the Buyer Acceptance process.
- Receive the signature2 from the Buyer Acceptance process.
- Invoke the Supplier process to send the signatures of the Buyer Acceptance process.

End sequence

4.4 Security Information:

Because messages can be modified, it is recommended that business process implementation use WS-Security (web service security). It provides security by keeping security information in the SOAP part of the message. WS-security does not provide fairness and accountability. To fulfill such requirements there is a need to use the fair non-repudiation protocol.

4.5 Petri Net Model of BPEL:

BPEL processes consist of two types, abstract process and executable process. Both processes contain elements that can be model in Petri nets.

4.5.1 Petri Net Model of Abstract BPEL Process:

In our model the mapping of WSDL's parts [12] is as below

Place → PortType (Operations – input, output messages)
Transition → ServiceLinkType (Name, my role, partner role)
Token → Message (Data)
Arc → Binding

Note: The Service link type definition can be placed within the WSDL document defining the portTypes from which the different roles are defined.

4.5.2 Petri Net Model of Executable BPEL Process:

We can model BPEL process using Petri net. The mapping for BPEL process is defined as:

Place → Containers
Transition → Invoke, Receive, Reply, Assign, Switch
Token → Message (Data)

A sequence activity is represented hierarchically and can be refined into a number of lower level activities such as invoke, receive etc.

Models of each process are merged to obtain a system-wide view a complete web business transaction. Although the individual models may display the desired properties of livness, safeness and complete termination, the merged net may not display such properties.

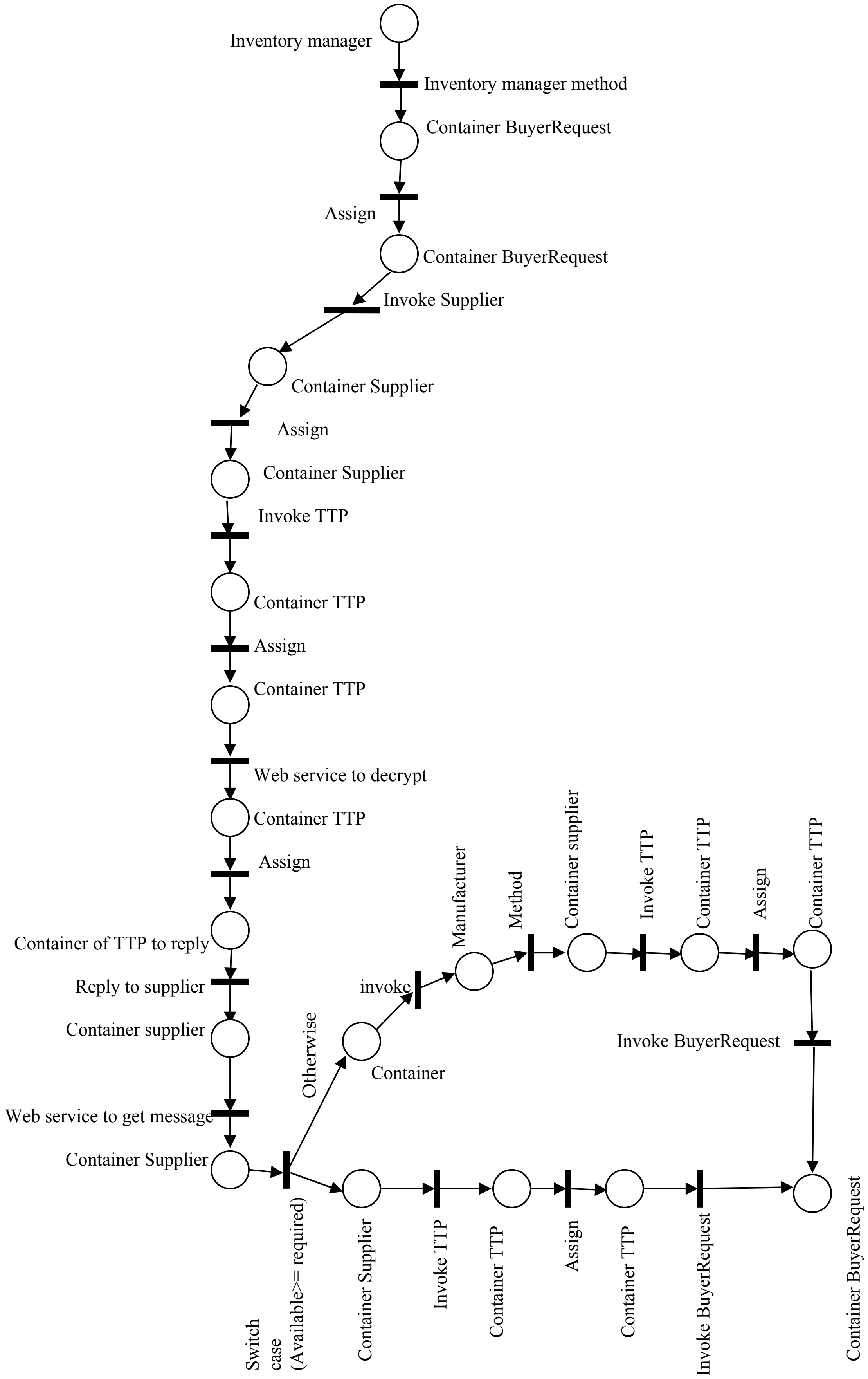
4.5.3 Petri Net Model of Paradigm:

The Petri net modeling captures websites, which use web services, such as the inventory manager's website in our case. The web service on that site calls the Buyer Request process which may invoke different processes to fulfill the request and so on. The whole procedure of business processes and transactions involving web services is represented as a big web.

To draw the Petri net of B2B processes, global information of the processes are required. Each process is only aware of itself and other web services (or BPEL processes) it calls. The entire business transaction can be therefore modeled by merging the models of individual transactions.

When a process or web service needs to be invoked, its respective WSDL file is traced. A WSDL file has all the information required to communicate. The complete Petri net represents all the possible execution paths of the whole system, in our case inventory manager's web method (web services) is followed by the WSDL and then the web service and then WSDL of process and so on.

Figure 4.6 shows the Petri net model of executable BPEL processes. It starts with inventory manager. This model also shows a switch statements in the supplier process when number of available products is less then the requested. It invokes manufacturer in order to fulfill the demand. Places represent containers in process, Transitions represents invoke and incoming Arcs to the transitions represent out going data and outgoing Arcs from the transitions is incoming data. Tokens are data.

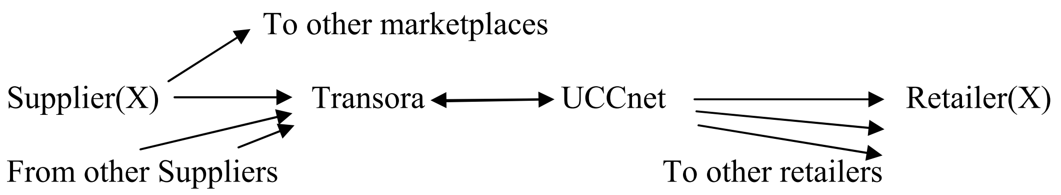


Inventory manager
Inventory manager method
Container BuyerRequest
Assign
Container BuyerRequest
Invoke Supplier
Container Supplier
Assign
Container Supplier
Invoke TTP
Container TTP
Assign
Container TTP
Web service to decrypt
Container TTP
Assign
Container of TTP to reply
Reply to supplier
Container supplier
Web service to get message
Container Supplier
Switch case (Available>= required)
Otherwise
Container
invoke
Manufacturer
Method
Container supplier
Invoke TTP
Container TTP
Assign
Container TTP
Invoke BuyerRequest
Container Supplier
Invoke TTP
Container TTP
Assign
Container TTP
Invoke BuyerRequest
Container BuyerRequest

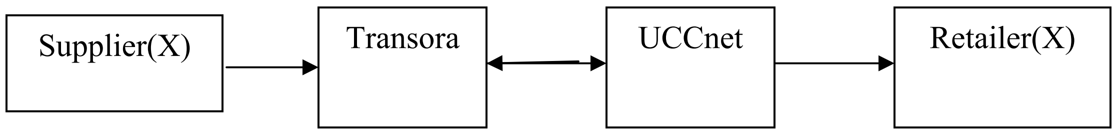
Figure 4.4: Petri net modeling of BPEL process

4.6 A Novel Non-repudiation Protocol for a Chain of Business Transactions:

The protocol presented in Section 4.2 established non-repudiation between two individual parties. However, business transactions are rarely so simple, and may involve more parties in many different topologies. For example, in a multicast scenario, a request (such as a contract bid) may be multicast to a number of different vendors. Each vendor may contact their suppliers for more information. Some, if not all of the vendors will come back with a quote to the originator of the request. There is a need for non-repudiation in such a multicast environment. It is beyond the scope of this thesis to consider all possible scenarios. We consider a chain-linked business transaction. Assume a supplier (X) wants to publish details about a new product (say a bar of soap). He publishes the information to a public Market Place called Transora. Transora gets the information from a lot of suppliers. Retailer (X) sells soap and wants to know when new soap products are available. Retailer (X) has relationship with UCCnet. UCCnet sends information to a lot of retailers. The flow will be:



We simplify above initially to just:

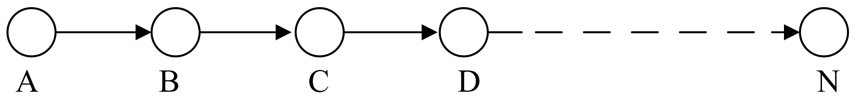


Issues:

How can Retailer(X) be guaranteed that the information he received is indeed from Supplier(X)? Or how can Supplier(X) be guaranteed that Retailer(X) did actually get the new product detail? We propose a novel non-repudiation protocol for chain linked business transactions.

Approach:

Non-repudiation in a chain linked system is as follow.



There are following cases

Case 1: $A \longrightarrow N$

Intermediate businesses (B, C, D …) cannot access message or key.

Case 2: $A \longrightarrow N$

Intermediate businesses (B, C, D …) can access message, modify it or add their own information.

4.6.1 Design and Approach; Case 1:

To avoid modification in the message by the intermediate nodes, there must be a Non-repudiation protocol down the whole business linked chain. The proposed protocol works as follow. We have borrowed notation from [15]

Where K: a symmetric key generated by A

t_{id}: transaction id

$em = eK(M)$

$ek\_from\_A = eP_N(K)$ $dek = eP_{TTP}(ek\_from\_A)$

$md1 = MD(em)$ $md2 = MD(dek)$ $md3 = MD(id\_N)$

$id\_N = eP_N(id\text{-}message)$ id-message generated by originator

$id\_TTP = eP_{TTP}(id\text{-}message)$

treble signature = $t_{id} \mid md1 \mid md2 \mid md3 \mid sS_A(t_{id} \mid md1 \mid md2 \mid md3)$

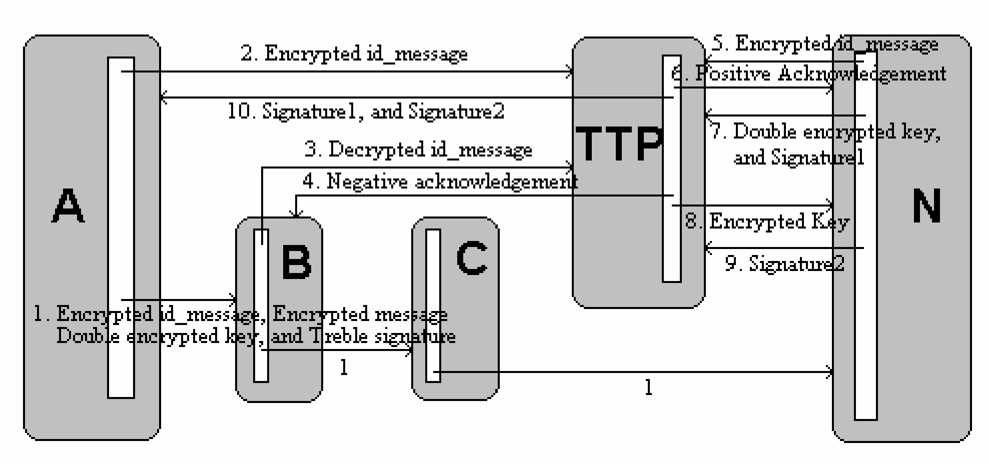

A
B
C
TTP
N
2. Encrypted id_message
10. Signature1, and Signature2
3. Decrypted id_message
4. Negative acknowledgement
5. Encrypted id_message
6. Positive Acknowledgement
7. Double encrypted key, and Signature1
8. Encrypted Key
9. Signature2
1. Encrypted id_message, Encrypted message Double encrypted key, and Treble signature
1
1

Figure 4.5: A Non-repudiation Protocol for Chained Linked Business Transaction.

Step 1: The A sends the encrypted id-message, encrypted message, double encrypted key and treble signature to the B, who sends it to the C and so on until it reaches the N.

Message from A $\longrightarrow$ B : $t_{id} \mid A \mid B \mid N \mid id\_N \mid em \mid dek \mid$ treble signature

Message from B $\longrightarrow$ C : $t_{id} \mid B \mid C \mid N \mid id\_N \mid em \mid dek \mid$ treble signature

Note: This will not stop intermediate nodes from corrupting the message.

Step 2: The A encrypts the id-message with public key of TTP and sends it to the TTP.

$$A \longrightarrow TTP : t_{id} \mid eP_{TTP}(id\text{-}message)$$

Step 3 : Now suppose an intermediate node B tries to get the key by sending the decrypted id_message $dS_B(id\_N')$ (it is not same as id_N).

$$B \longrightarrow TTP : t_{id} \mid dS_B(id\_N')$$

The TTP will not accept because id_message is not equal to id_message he received in step 2 from A.

Step 4: TTP $\longrightarrow$ B : t_{id} | Negative acknowledgement

Now consider messages with the recipient N. First the recipient N needs to identify itself.

Step 5: id_N is first decrypted at the N using private key of the N: dS_N(id_N). It is next encrypted using public key of the TTP and sent to the TTP.

$$N \longrightarrow TTP : t_{id} \mid eP_{TTP}(id\_message)$$

Step 6: $TTP \longrightarrow N$: Positive acknowledgement

Step 7: The recipient N sends double encrypted key and signature1 to the TTP

$$N \longrightarrow TTP : t_{id} \mid A \mid N \mid md1 \mid md3 \mid dek \mid sS_N(t_{id} \mid md1 \mid md3)$$

Step 8: TTP decrypts the double encrypted key and sends encrypted key to the recipient N.

$$TTP \longrightarrow N : t_{id} \mid ek\_from\_TTP$$

Where, ek_from_TTP = dS_{TTP}(dek) : decryption of dek using private key of TTP.

Step 9: The recipient N sends his signature2 on a digested secret key to the TTP.

$$N \longrightarrow TTP : t_{id} \mid sS_N(MD(ek\_from\_TTP))$$

Step 10: TTP sends both signatures to the originator A.

$$TTP \longrightarrow A : t_{id} \mid sS_N(t_{id} \mid md1 \mid md3\ ) \mid sS_N(MD(ek\_from\_TTP))$$

Requirements:

In this section we give an informal analysis on how our protocol satisfies the requirements of a chain of Business Transactions.

Requirement 1: Intermediate nodes cannot get the key from the TTP because of id_N.

Requirement 2: How does the originator A know that the recipient N gets message? It is because of md3 in the recipient N's signature.

Requirement 3: How does the originator A know that message is correct? It is because of md1 in the recipient N's signature.

Requirement 4: How does the originator A know that the key is delivered correctly? It is because of the signature of the recipient N on the digested secret key i.e. sS_N(MD(ek_from_TTP)) .

Requirement 5: How does N know that this message is from the originator A? The recipient N checks the integrity of message using treble signature. It is the only sender that can generate that signature.

4.6.2 Petri Net Model of Non-repudiation chain-linked Protocol:

The protocol is modeled in figure 4.4. It started with the place p_1 and ended with the final place p_{12}. Each process task is represented by a transition [18]. There is a token in p_1 thus t_1, which represents process A, is the only enabled transition. Firing t_1 consumes the tokens in p_1 and produced tokens in p_2 and p_6 then t_2, which represents process B, is enabled. Transition t_5 is not enabled at this time because it required the token in p_5. Firing t_2 consumes the tokens in p_2 and produced new tokens in p_3 and p_{13}, and then t_3 and t_{11} are enabled. Transition t_{12} represents the negative acknowledgment from the TTP to the process B when it tried to access message. Now t_3, which is process C, is enabled. Firing t_3 consumes tokens in p_3 and produced tokens in p_4. This process continues until message reaches the recipient N. Figure 4.4 shows the process N as a transition t_4 and is enabled as p_4 contain tokens. On firing t_4 it produced tokens in p_5. Now p_6 and p_5 contain tokens so transition t_5, which represents process TTP, is enabled. It guarantees the correct execution of parallel tasks [18]. If t_5 fires, both tokens in p_6 and p_5 are consumed and a token is produced in p_7. Now transition t_6 is enabled. On firing, it consumed tokens from p_7, which is a positive acknowledgment from the TTP process to the recipient N, and tokens are produced in p_8. This process continued until t_{10} is fired, which is process A, to get both signatures from TTP and produced tokens in p_{12}. It means one instances of message passage from the process A to the Process N has finished.

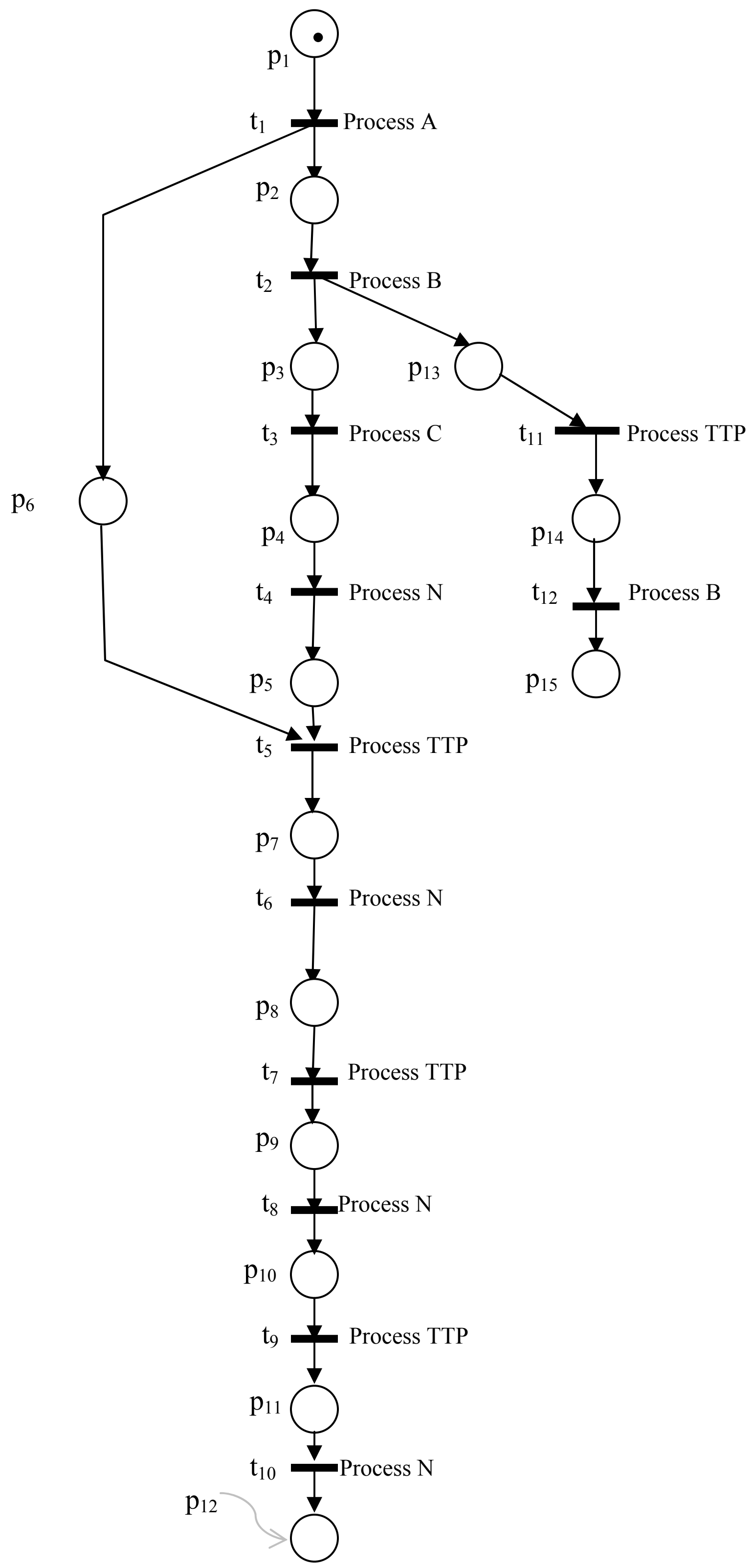


Figure 4.6: Case1; Petri Net Model of Chain Linked Business Transaction

4.6.3 Dispute Resolution:

Two kinds of disputes can come up [23]: repudiation of origin and repudiation of recipient. Repudiation of recipient arises when the originator A claims having sent a message to the recipient N, who denies having received it. Repudiation of origin arises when the recipient N claims having received the message from the originator A, who denies having sent it.

Repudiation of Recipient:

If the recipient N denies receiving message 'M', the originator A can present evidence in the form of signatures of N plus (t_{id}, em, dek, id_message, md1, md2, md3, K, M, P_{TTP}, P_N, ek_from_TTP) to arbitrator. The arbitrator will compare the t_{id} and check

em = eK(M)

md1= MD(em)

dek = eP_{TTP}(ek_from_A)

ek_from_A =eP_N(K)

md2= MD(dek)

md3=MD(id_N)

Message digest of ek_from_TTP

N's signatures $sS_N(t_{id} \mid md1 \mid md3)$, and sS_N(MD(ek_from_TTP))

TTP's signature dS_{TTP}(dek) and its log information to get signature.

The originator A will win the dispute if all the checks are positive. Originator A will win even if he is unable to provide log information of the TTP as in last check. So it is not required the presence of the TTP at the time of dispute.

Repudiation of Origin:

If the originator A denies sending the message 'M', the recipient N can present evidence in the form of treble signature of A plus (t_{id}, em, dek, md1, md2, md3, K, M, ek_from_TTP) to the arbitrator. The arbitrator will compare the t_{id} and check

$$em = eK(M)$$

$$md1 = MD(em)$$

$$dek = eP_{TTP}(ek\_from\_A)$$

$$ek\_from\_A = eP_N(K)$$

$$md2 = MD(dek)$$

$$md3 = MD(id\_N)$$

$$\text{A's signature } sS_A(t_{id} \mid md1 \mid md2 \mid md3)$$

$$\text{TTP's signature } dS_{TTP}(dek)$$

Recipient N will win the dispute if all the checks are positive.

4.6.4 Security Requirements:

Important requirements of non-repudiation services are fairness, timeliness, protection and confidentiality.

Fairness of Protocol:

A non-repudiation protocol provides fairness if neither party can gain an advantage by quitting premature or misbehaving during the execution of protocol [23]. If the protocol terminates at step 1 because of some problem in communication channel or misbehavior of the intermediate node, the originator loss nothing. At this time, even intermediate nodes or the recipient N has an encrypted message em = eK(M) and double encrypted key dek = eP_{TTP}(ek_from_S) but they cannot access the message until the TTP decrypts the key. If any intermediate node tries to access secret key, first he needs to identify himself by decrypting id_N and this is not possible because id_message is encrypted with public key of recipient N.

The recipient N gets access to entire original message after step 8. After this step the recipient N can misbehave in two ways. Recipient N does not execute step 9. TTP detects the misbehavior of the receiver when timeout is reached and TTP does not receive N's signature. In this case TTP sends $sS_N(t_{id}$ | md1 | md3) and ek_from_TTP to the originator A. Originator A proves the misbehavior by presenting t_{id}, em, id_message and dek and showing that em and id_message are matched to the message digest in $sS_N(t_{id}$ | md1 | md3) and dek is matched to the ek_from_TTP.

Recipient N can also misbehave by deliberately signing on a fake key in step 9 to reject the transaction later. In this case the originator A can prove the dishonesty of the recipient by showing that key sent at step 1 matched with ek_from_TTP at step 8 but not to the signature2 of the N in step 9.

The protocol also prevents the originator A from sending an invalid message or denying sending a message. In the final step, the originator A can get a receipt in the form of digital signature. However, final step cannot be reached if the originator A has sent an invalid message because the receiver N can check the integrity of message by comparing the message digest of the encrypted message and double encrypted key with the treble signature. If the recipient N did not receive the encrypted message and double encrypted key correctly, he would not give the signature1. Recipient N would not send the second signature if he cannot access the encrypted message using the secret key. In court, N can presents that key and message digest of the key that he received in treble signature and showing that the key corresponds to the its message digest but unable to unlock the message. There is no way that the originator A can denies having sent message and double encrypted key because of the treble signature. If originator A denies from these encrypted message and double encrypted key and claims that he sent different message and double encrypted key (which is different from original encrypted message and double encrypted message received by N), the recipient N can clear his position and disprove that claim by showing treble signature on message digest of em, dek and id_N that has been received.

Consider the situation when the TTP and the originator team up to get the recipient N in trouble. On receiving the double encrypted key, TTP does not send the encrypted key to the receiver and claims that he did. Now the recipient already has the treble signature so that he can have partial evidence on the double encrypted key. TTP cannot argue that he gave the encrypted key to the recipient, because the recipient would not give the second signature until he gets the encrypted key.

Protection and integrity of Message:

This protocol protects the involved parties from common message protection threats such as message interception and modification, and reply attacks. We used message digest and encryption techniques to protect the message from interception and modification. The integrity of message can be verified by comparing with the message digest value. Protection is provided by encrypting a message with a key which is double encrypted so that no one but recipient can access the message content. The protocol generates a new transaction id (t_{id}) every time to protect from replay attacks [15].

Confidentiality of transaction:

The protocol provides the confidentiality so that, except the recipient no one else including the TTP can access the original contents of the message. The recipient needs to identify itself by sending id_message to TTP. Although the intermediate nodes are involved in the communication, they cannot access the message. The only way to read the message is through the secret key that encrypts the message. The secret key is double encrypted to prevent the intermediaries and TTP from getting access to the key and hence the original message. In step 9 the recipient signs the message digest of the secret key but not the secret key itself so that TTP does not know the key, even though it is involved in the protocol.

Timeliness:

The Protocol achieves timeliness as each involved party can terminate the protocol at any time at their own judgment while maintaining fairness [23]. If the protocol terminates after step 1, the recipient N cannot take any advantage because it cannot access the message even it gets the treble signature. If it terminates at step 2 and step 3 or step 4 and step 5, nobody can claim any thing because these are only identification messages. If it ends at step 7, the originator A cannot claim anything because the recipient N has yet to sign on the digested secret key. If the protocol ends at step 8, the recipient N cannot take any advantage because of his signature over the encrypted message and log information over the key are available to the TTP when executing step 7. If protocol terminates at step 9 and before step 10, the originator A can contact the TTP later to get both signatures.

4.6.5 Building the BPEL Processes; Case 1:

For the BPEL specification we consider only one intermediate node in the above protocol. For more intermediate nodes, the BEPL specification for the intermediate nodes can be simply replicated for each intermediate node. . We therefore need only four processes, one for supplier A, one for buyer N, one for intermediate party B and one for TTP.

Supplier Process A:

This process takes the request from the process B and sends request with double-encrypted key to the intermediate process. The process A also sends id_message to the process TTP. Finally it receives signatures from the TTP process. So there are three partners in this transaction - Supplier A, intermediate node B and TTP. First we define partners and containers to store data.

After defining the partners and containers of the process, we need to define activities of the process starting from sequence of process.

Begin sequence

- Receive a request from the intermediate node.
- Invoke a process to produce encrypted id_message, encrypted message and double encrypted key.
- Assign the data to the container to be sent to intermediate node.
- Reply the intermediate node to send message.
- Assign encrypted id_message to a container.
- Invoke the process TTP and send encrypted id_message.
- Receive the signatures from the trusted third party that buyer access the original message.

End sequence

Intermediate Process B:

The sequence of activities at intermediate process is as follow:

Begin sequence

- Invoke the buyer process to send request of purchase.
- Receive the information from supplier process.
- Invoke the buyer process to send that information.

End sequence

Buyer Process N:

The sequence of activities is as follow:

- Receive the information from intermediate node.
- Assign the id_message to the container to be use in the invoke activity.
- Invoke the process TTP and send the id_message.
- Receive the acknowledgement from the process TTP.
- Assign double encrypted key and signature1 to the container.
- Reply to the process TTP and send double encrypted key and signature1.
- Receive encrypted key from the process TTP.
- Reply the process TTP and send the signature2.

End sequence

Process TTP:

The fourth BPEL process place at the TTP and its sequence of activities is as follow:

Begin sequence

- Receive id_message from the supplier process A.
- Receive id_message from the buyer process N.
- Invoke internal process to compare id_message for identification.
- Reply with the acknowledgement to the buyer process N.
- Receive double encrypted key and signature1 from buyer process.
- Reply with encrypted key to the buyer process.
- Receive the signature2 from the buyer process.
- Reply the supplier process to send the signatures of the buyer process.

End sequence

4.7 Chain linked Non-repudiation protocol for Modified Messages; Case 2:

The protocol presented in Section 4.7 established non-repudiation in chain linked business transactions where intermediate node cannot access and modify the messages. However, in some scenarios there are some transactions in which intermediate nodes may need to add their own information to the message.

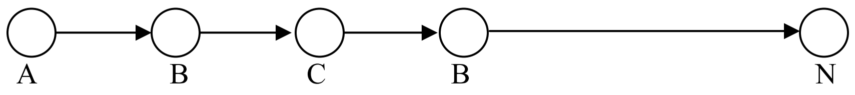
Example Scenario:

A given product is created by merging two products from different producers A and C. Producer A is sent the purchase order from the buyer N.

- Producer A accepts the order and sends the order fulfillment note to the buyer's preferred shipper B. Producer A has included the price in the order fulfillment note and does not want anyone else to know this price. It is also critical that no one else modify this price. Included is the number of units to be produced, shipping data of products (size, weight etc) and dates for shipment.

- Shipper B read his own contents e.g. shipping date, weight, and size etc. and sent information (including shipping) to the producer C.
- Producer C sends a note to the shipper B on when the complete order will be available and other information. Included also is the price that only Buyer N should read.
- Shipper B sends a note to the buyer N with price to ship final product to the buyer from the producer C.
- Buyer N receives order fulfillment note.

Flow for this chain linked scenario is as follow.



Issues:

- How does the recipient N know that intermediate node access only those pieces of the message that they should modify?
- How does N know that intermediate nodes did modify the pieces they were supposed to change?
- How does A know that message did get to the recipient N and intermediate nodes did make appropriate changes? etc.

We propose a new non-repudiation protocol for chain linked business transactions in which intermediate nodes can add their information in the message.

4.7.1 Design and Approach:

We divide the message into two different portions.

1. Segment that cannot be seen or modified by any intermediate node e.g. price of product from any producer. We encrypt this segment of message with a double encrypted key.
2. Segment that is viewable to the particular intermediate node e.g. shipping date, size and weight etc. We encrypt this segment of message with public key of particular intermediate node.

   The protocol works as follow.

Where K : a symmetric key generated by A

t_{id} : transaction id

$em(A) = eK(m_A)$

$ek\_from\_A = eP_N(K_A)$ $dek(A) = eP_{TTP}(ek\_from\_A)$

$md1 = MD(em(A))$ $md2 = MD(dek(A))$ $md3 = MD(id\_N)$

$md4=MD(em(C))$ $md5=MD(em(B))$

$id\_N = eP_N(id\text{-}message)$ $em\_for\_B = eP_B(M)$

$id\_TTP = eP_{TTP}(id\text{-}message)$

treble signature = $t_{id} \mid md1 \mid md2 \mid md3 \mid sS_A(t_{id} \mid md1 \mid md2 \mid md3)$

Step 1: The originator A sends encrypted id-message, encrypted message (this segment is encrypted with a double encrypted key), double encrypted key, treble signature and encrypted message to B (this segment is encrypted with a public key of B).

$A \longrightarrow B$: t_{id} | A | B | N | id_N | em(A) | dek(A) | treble signature | em_for_B

Step 2: The originator A encrypts the id-message with public key of the TTP and sends it to the TTP.

$A \longrightarrow TTP$: $t_{id} \mid eP_{TTP}$ (id-message)

Step 3: The intermediate node B decrypts message M using its private key and sent an encrypted message (encrypted with a public key of C) to the Producer C.

$$B \rightarrow C: t_{id} \mid eP_C(message)$$

Step 4: The producer C gets the information and sends an encrypted message, double encrypted key, dual signature and encrypted message (encrypted with a public key of B) to the shipper B.

$em(C) = eK(m_C)$ $ek\_from\_C = eP_N(K_C)$ $dek(C) = eP_{TTP}(ek\_from\_C)$

$$C \rightarrow B : t_{id} \mid C \mid B \mid N \mid em\_for\_B \mid em(C) \mid dek(C) \mid Dual\_signature\_from\_C$$

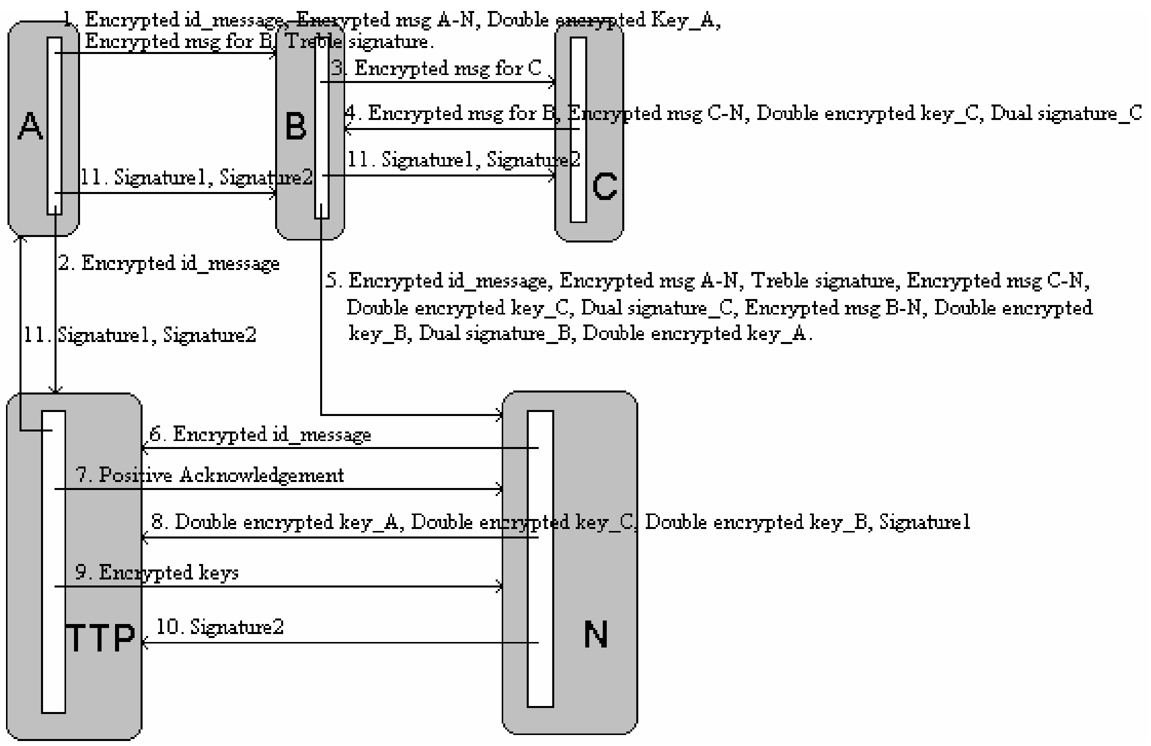

1. Encrypted id_message, Encrypted msg A-N, Double encrypted Key_A, Encrypted msg for B, Treble signature.
3. Encrypted msg for C
4. Encrypted msg for B, Encrypted msg C-N, Double encrypted key_C, Dual signature_C
11. Signature1, Signature2
11. Signature1, Signature2
A
B
C
2. Encrypted id_message
11. Signature1, Signature2
5. Encrypted id_message, Encrypted msg A-N, Treble signature, Encrypted msg C-N, Double encrypted key_C, Dual signature_C, Encrypted msg B-N, Double encrypted key_B, Dual signature_B, Double encrypted key_A.
6. Encrypted id_message
7. Positive Acknowledgement
8. Double encrypted key_A, Double encrypted key_C, Double encrypted key_B, Signature1
9. Encrypted keys
10. Signature2
TTP
N

Figure 4.7: Non-repudiation protocol for chain-linked business transaction with message modification.

Step 5: The shipper B gets the information and forwards message from A (encrypted message , double encrypted key, id-message and treble signature), message from C (encrypted message, double encrypted key and dual signature), and sends his own encrypted message, double encrypted key and dual signature to the Buyer N.

$em(B) = eK(m_B)$ $ek\_from\_B = eP_N(K_B)$ $dek(B) = eP_{TTP}(ek\_from\_B)$

$$B \longrightarrow N: t_{id} \mid B \mid N \mid em(A) \mid dek(A) \mid \text{treble signature} \mid em(C) \mid dek(C) \mid id\_N \mid$$
$$Dual\_signature\_from\_C \mid em(B) \mid dek(B) \mid Dual\_signature\_from\_B$$

Step 6: id_N is first decrypted at N using private key of N: $dS_N(id\_N)$. It is next encrypted using public key of TTP and sent to the TTP.

$$N \longrightarrow TTP : t_{id} \mid eP_{TTP}(\ id\_message)$$

Step 7: $TTP \longrightarrow N : t_{id} \mid$ Positive acknowledgement.

Step 8: The recipient N sends Signature1, double encrypted key from A, double encrypted key from C, and double encrypted key from B to the TTP.

$$N \longrightarrow TTP : t_{id} \mid A \mid N \mid md1 \mid md3 \mid md4 \mid md5 \mid dek(A) \mid dek(C) \mid dek(B) \mid sS_N(t_{id} \mid$$
$$md1 \mid md3 \mid md4 \mid md5)$$

Step 9: TTP decrypts the double encrypted keys and sends the encrypted keys to N.

$$TTP \longrightarrow N : t_{id} \mid eks\_from\_TTP$$

Step 10: The recipient N sends its signature2 on the digested secret keys to TTP.

$$N \longrightarrow TTP : t_{id} \mid sS_N(MD(eks\_from\_TTP))$$

Step 11: TTP forwards both signatures to the producer A which then forward it to the shipper B and producer C.

$$TTP \longrightarrow A : t_{id} \mid sS_N(t\ id \mid md1 \mid md3 \mid md4 \mid md5\ ) \mid sS_N(MD(eks\_from\_TTP))$$

Requirements:

In this section we give an informal analysis on how our protocol satisfies the requirements for a chain of Business Transactions.

Requirement 1: How can the buyer N be guaranteed that the price it gets is indeed what was input by producer A (or C)?Buyer N checks the integrity of message from A using treble signature and from C and B using their dual signature. These are the only senders that can generate these signatures.

Requirement 2: Intermediate node cannot get the key and hence the message from TTP because of eP_N(id_message).

Requirement 3: How can the A, B and C are guaranteed that buyer N did get the correct information and it was not tampered or read by any of the intermediate notes? It is because of md1, md4 and md5 respectively in N's signature.

Requirement 4: How can the producer A be guaranteed that the information was not transmitted to a potential competitor (e.g. Producer C may have a preferred partner and may send some of the product details to this partner)? A encrypts the confidential information with a double encrypted key and no one can get access to key without identifying himself.

Color Petri Net:

Definition: A colored Petri Net (CPN) is a tuple CPN = (PN, Σ, CR, E) [20] where

1. $PN = (P, N, F, M)$ in an ordinary Petri net,
2. $\Sigma = \{\sigma_1, \sigma_2, ...\}$ is a finite set of colors,
3. CR is color factor such that $CR(p) \subseteq \Sigma$, and $CR(m(p)) \subseteq CR(P)$ (see glossary for more explanation), and
4. E, the arc function such that: $\forall f(p, t), f(t, p) \in F, E_f \subseteq CR(p)_{MS}$ (see glossary for more explanation)

4.7.2 Color Petri Net Based Model of Case 2:

We prefer color Petri Net to model Non-repudiation protocol as intermediate node can modify messages and it is simple to flow the each message along the whole chain.

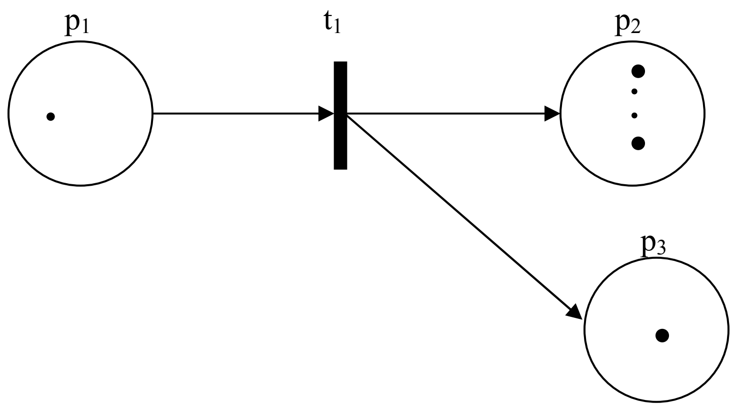
Buyer N sent the purchase order to the Producer A, so token with color 'a' starts instance in place p_1.

Place → Process

Transition → Method

Token → Data

Step1:



$C(p_1) = a$ $C(p_2) = \{b, c, e, f, g\}$ $C(p_3) = d$

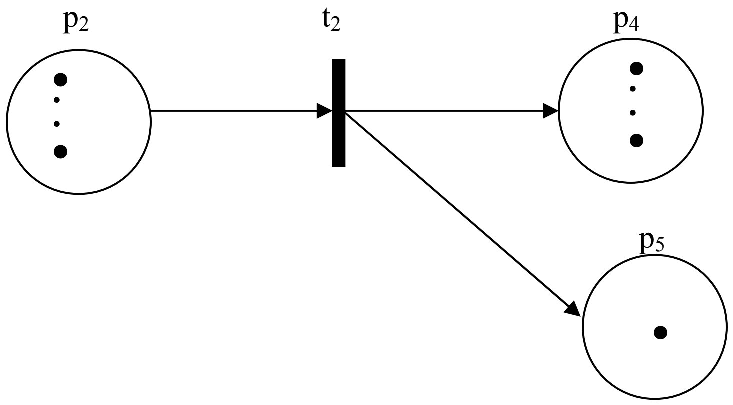
$E_{f(p1,t1)} = a$ $E_{f(t1,p2)} = b + c + e + g + f$ $E_{f(t1,p3)} = d$

Where, a = start instance b = dek(A) c = em_for_B d = id_TTP

e = id_N f = treble signature g = em(A)

Consider the CPN, there are three places p_1, p_2 and p_3 and a transition t_1 such that the color set of p_1, p_2 and p_3 are $CR(p_1) = \{a\}$, $CR(p_2) = \{b, c, e, g, f\}$, and $CR(p_3) = \{d\}$, respectively. So p_1 is initially marked with token 'a' to represent token of color 'a'. The arc function associated with the three arcs *f (p1, t1), f (t1, p2), and f (t1, p3)* are $E_{f(p1,\, t1)} = a$, $E_{f(t1,\, p2)} = b + c + e + g + f$, and $E_{f(\,t1,\, p3)} = d$, respectively, meaning that enabling t_1, which represents a method at process A, needs a token of color ‘a’ and firing t_1 produces tokens of color b, c, e, f, and g in place p_2 and token of color ‘d’ in place p_3. Since $E_{f(p1, t1)} \leq m(p_1)$ (i.e. a $\leq$ a) so t_1 is enabled. We are following the notations as in [20]

Step2:



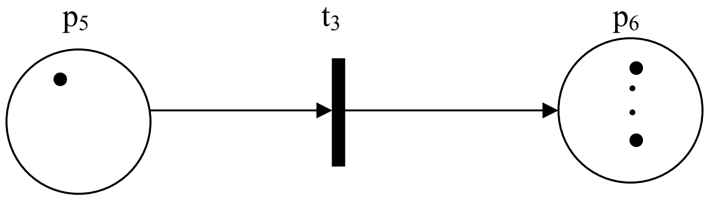
$C(p_2) = \{b, c, e, f, g\}$ $C(p_4) = \{b, e, f, g\}$ $C(p_5) = h$

$E_{f(p2, t2)} = b + c + e + f + g$ $E_{f(t2, p4)} = b + e + f + g$ $E_{f(t2, p5)} = h$

Where, h = ePc(message)

Since $E_{f(p2, t2)} \leq m(p_2)$ (i.e. $b + c + e + f + g \leq b + c + e + f + g$) so t_2 is enabled. Firing t_2, which represents a method at process B, produces tokens of color b, e, f, and g in place p_4 and token of color ‘h’ in place p_5.

Step3:



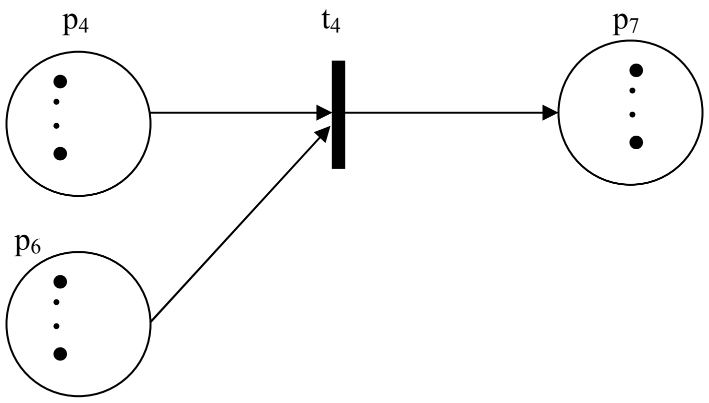
$C(p_5) = h$ $C(p_6) = \{o, q, r, s\}$

$E_{f(p5, t3)} = h$ $E_{f(t3, p6)} = o + q + r + s$

Where, o = em(C) q = dek(C) r = Dual_signature_from_C s = em_for_B

Since $E_{f(p5,\ t3)} \leq m(p_5)$ (i.e. h $\leq$ h) so t_3 is enabled. Firing t_3, which represents a method at process C, produces tokens of color o, q, r, and s in place p_6.

Step 4:



$C(p_4) = \{b, e, f, g\}$ $C(p_6) = \{o, q, r, s\}$ $C(p_7) = \{e, b, g, f, k, i, n, o, q, r\}$

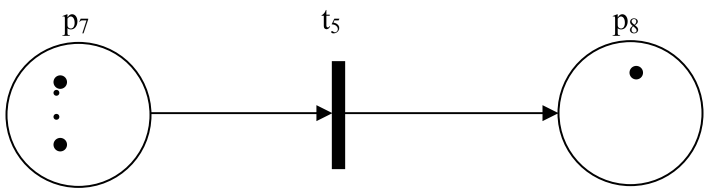
$E_{f(p4,\ t4)} = b + e + g + f$ $E_{f(p6,\ t4)} = o + q + r + s$

$E_{f(t4,\ p7)} = b + e + f + k + i + n + o + q + r$

Where, k = em(B) i = dek(B) n = Dual_signature_from_B

Since $E_{f(p4,\ t4)} \leq m(p_4)$ and $E_{f(p6,\ t4)} \leq m(p_6)$ so t_4 is enabled. Firing t_4, which represents a method at process B, produces tokens of color e, b, g, f, k, i, n, o, q, and r in place p_7.

Step5:



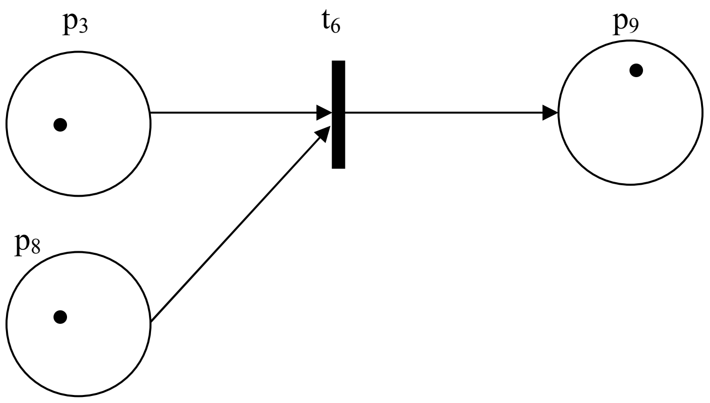
$C(p_7) = \{e, b, g, f, k, i, n, o, q, r\}$ $C(p_8) = u$

$E_{f(p7,\ t5)} = e$ $E_{f(t5,\ p8)} = u$

Where, $u = eP_{TTP}$(id_message)

Since $E_{f(p7,\, t5)} \leq m(p_7)$ (i.e. $e \leq e + b + g + f + k + i + n + o + q + r$) so t_5 is enabled. Firing t_5, which represents a method at process N, produces a token of color ‘u’ in place p_8.

Step 6:



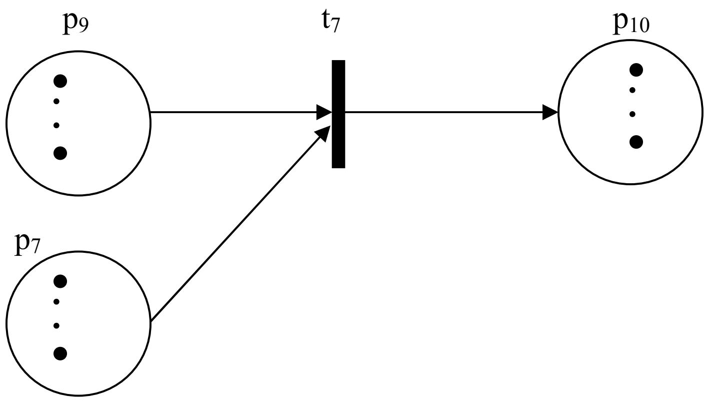
$C(p_3) = d$ $C(p_8) = u$ $C(p_9) = v$

$E_{f(p3,\, t6)} = d$ $E_{f(p8,\, t6)} = u$ $E_{f(t6,\, p9)} = v$

Where, v = positive acknowledgement from the TTP

Since $E_{f(p3,\, t6)} \leq m(p_3)$ and $E_{f(p8,\, t6)} \leq m(p_8)$ so t_6 is enabled. Firing t_6, which represents a method at process TTP, produces a token of color ‘v’ in place p_9.

Step 7:



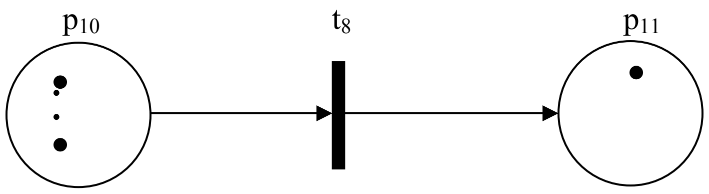
$C(p_9) = v$ $C(p_7) = \{b, g, f, k, i, n, o, q, r\}$ $C(p_{10}) = \{b, q, i, w\}$

$E_{f(p9,t7)} = v$ $E_{f(p7,t7)} = b + q + i$ $E_{f(t7,p10)} = b + q + i + w$

Where, $w = sS_N(\text{t id} \mid md1 \mid md3 \mid md4 \mid md5)$

Since $E_{f(p7,t7)} \leq m(p_7)$ and $E_{f(p9,t7)} \leq m(p_9)$ so t_7 is enabled. Firing t_7, which represents a method at process N, produces tokens of color b, q, i, and w in place p_{10}.

Step8:



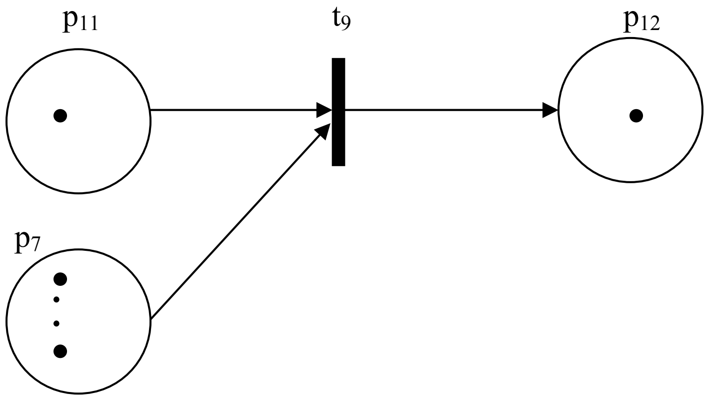
$C(p_{10}) = \{b, q, i, w\}$ $C(p_{11}) = x$

$E_{f(p10,t8)} = b + q + i$ $E_{f(t8,p11)} = x$

Where,   u = eks_from_TTP

Since $E_{f(p10,t8)} \leq m(p_{10})$ (i.e. $b + q + i \leq b + i + q + w$) so t_8 is enabled. Firing t_8, which represents a method at process TTP, produces a token of color 'x' in place p_{11}.

Step 9:



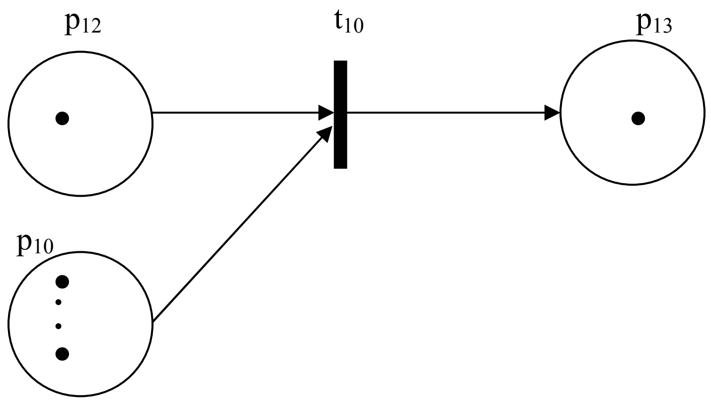
$C(p_{11}) = x$ $C(p_7) = \{g, f, k, n, o, r\}$ $C(p_{12}) = y$

$E_{f(p7,t9)} = g + k + o$ $E_{f(p11,t9)} = x$ $E_{f(t9,p12)} = y$

Where, $y = sS_N(MD(\text{eks\_from\_TTP}))$

Since $E_{f(p11,\, t9)} \leq m(p_{11})$ and $E_{f(p7,\, t9)} \leq m(p_7)$ (i.e. g + k + o $\leq$ g + f + k + n + o + r) so t_9 is enabled. Firing t_9, which represents a method at process N, produces tokens of color 'y' in place p_{10}.

Step 10:



$C(p_{12}) = y$ $C(p_{10}) = w$ $C(p_{13}) = \{w, y\}$

$E_{f(p12,\, t10)} = y$ $E_{f(p10,\, t10)} = w$ $E_{f(t10,\, p13)} = w + y$

Since $E_{f(p12,\, t10)} \leq m(p_{12})$ (i.e. $y \leq y$) and $E_{f(p10,\, t10)} \leq m(p_{10})$ (i.e. $w \leq w$) so t_{10} is enabled. Firing t_{10}, which represents a method at process TTP, produces tokens of color w and y in place p_{10}.

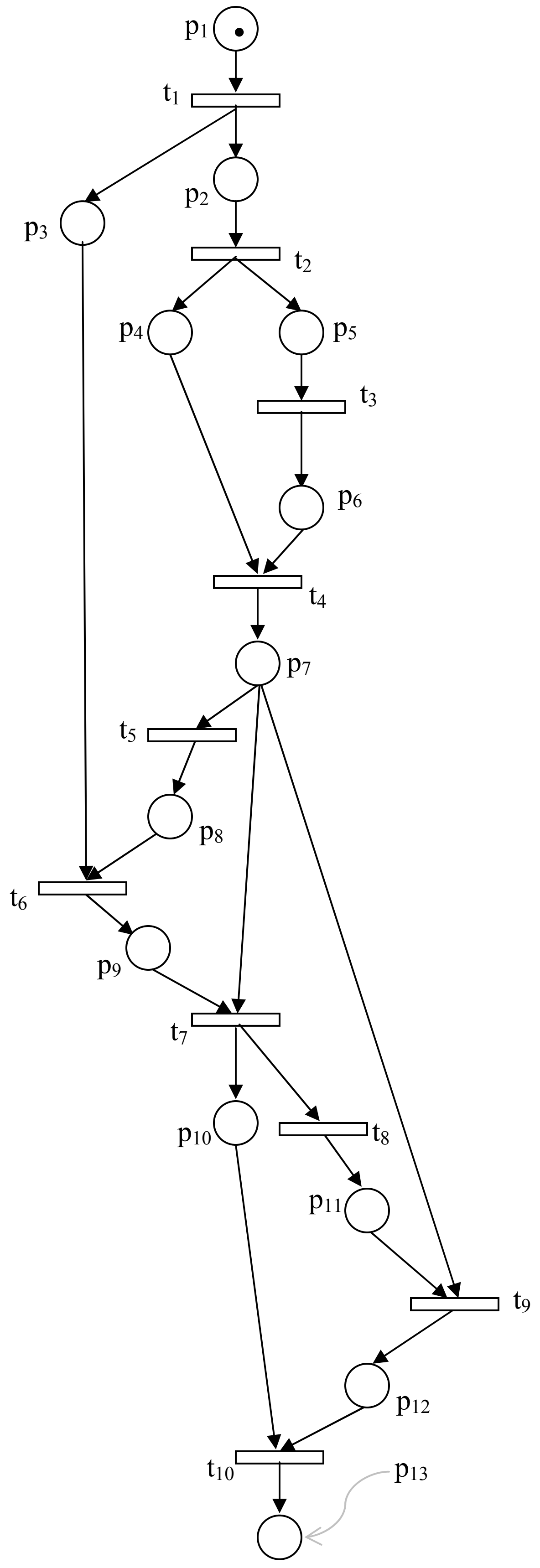

p1
t1
p2
p3
t2
p4
p5
t3
p6
t4
p7
t5
p8
t6
p9
t7
p10
t8
p11
t9
p12
t10
p13

Figure 4.8: Case2; Color Petri Net Model of Chain Linked Business Transaction

4.7.3 Reliability of Non-repudiation Protocol for Chain-Linked Transactions:

In this section, we show that if any transaction does not take place due to communication failures or node misbehavior, the protocol will terminate.

Definition 1:

Given a CPN, we define the number of distinct color associated with a place p_i as $u_i = | C(p_i) |$. [20]

Definition 2:

Given a CPN, we define the number of ways in which a transition t_i can fire as v_i = the number of consistent substitutions of each arc function $f(p_j, t_i)$ (the condition to be satisfied for the transition to fire) with the elements in $C(p_j)$, where $p_j \in \bullet t_i$. [20]

We can regard a colored Petri net as continuous time homogeneous Markov process and we can analyze the system reliability by means of analytic method [22].

Definition 3:

System is reliable if and only if each input and output function of all transitions are reliable. Where, reliability of the system is denoted by R(system)

$$R(\text{system}) = R\ (I(t_j))\ \text{AND}\ R\ (O(t_j))$$

$$R(\text{system}) = R\ (f(p_i, t_j))\ \text{AND}\ R(f(t_j, p_k))$$

Now first consider $R\ (f(p_i, t_j))$, where $p_i \in \bullet t_j$ (set of input places of t_j)

Unfolding the CPN as follow [20]:

For each place p_i in CPN, create as many places as u_i and label them with color $\sigma_1, \sigma_2, \sigma_3 \ldots\ldots, \sigma_u$ and for each transition t_j in CPN create as many transitions as v_j and give them distinct label to each.

Now draw the edges from every place derived from p_i to every transition t_j with arc function $E_{f\ (pi,\ tj)}$ and substitute σ_k in $E_{f\ (pi,\ tj)}$ with logical 1 which ensure a correct execution of t_j, so

$$R\ (f(p_i, t_j)) = \prod_{i=1}^{u} E_{f\ (pi,\ tj)}$$

Now consider $R(f(t_j, p_k))$, where $p_k \in t_j\bullet$ (set of output places of t_j)

For each place p_k in CPN, create as many places as u_k and label them with color $\sigma_1, \sigma_2, \sigma_3 \ldots\ldots, \sigma_k$ and for each transition t_j in CPN create as many transitions as v_j and give them distinct label.

Now draw the edges from every transition derived from t_j to every transition p_k with arc function $E_{f\ (tj,\ pk)}$ and substitute σk in $E_{f\ (tj,\ pk)}$ with logical 1 which ensure a correct execution of tj, so

$$R\ (f(t_j, p_k)) = \prod_{i=1}^{u} E_{f\ (pi,\ tj)}$$

Hence;

$$R(\text{system}) = R\ (f(p_i, t_j))\ \text{AND}\ R(f(t_j, p_k))$$

This shows that in the colored Petri net a transition may not fire properly (due to communication failure or misbehaving nodes). We assume that the Petri Net is live. If a transition does not fire, then the liveness property is no longer true and this will terminate the system.

4.7.4 Dispute Resolution:

There are two kinds of disputes in this protocol: repudiation of origin and repudiation of receipt. Repudiation of receipt arises when the senders claim having sent a message to the recipient N, who denies having received it. Repudiation of origin arises when the recipient N claims having received a message(s) from the sender(s), who denies having sent the message(s).

Repudiation of Recipient:

If the recipient N denies receiving messages m_A, m_B, and m_C, entity A, B and C can present evidence in the form of signatures of N plus (t_{id}, em(A), em(B), em(C), dek(A), dek(B), dek(C), id_message, md1, md3, md4, md5, K, m_A, m_B, m_C, P_{TTP}, P_N, eks_from_TTP) to arbitrator. The arbitrator will compare the t_{id} and check

$$em(A) = eK(m_A) \qquad em(B) = eK(m_B) \qquad em(C) = eK(m_C)$$

$$md1 = MD(em(A)) \qquad md3 = MD(em(id\_N)) \qquad md4 = MD(em(B))$$

$$md5 = MD(em(C))$$

$$ek\_from\_A = eP_N(K_A) \qquad dek(A) = eP_{TTP}(ek\_from\_A)$$

$$ek\_from\_C = eP_N(K_C) \qquad dek(C) = eP_{TTP}(ek\_from\_C)$$

$$ek\_from\_B = eP_N(K_B) \qquad dek(B) = eP_{TTP}(ek\_from\_B)$$

Message digest of eks_from_TTP

N's signatures $sS_N(t_{id} \mid md1 \mid md3 \mid md4 \mid md5)$, and $sS_N(MD(ek\_from\_TTP))$

TTP's signatures $dS_{TTP}(dek(A))$, $dS_{TTP}(dek(B))$ and $dS_{TTP}(dek(C))$,and its log information to get signature.

Senders A, B and C will win the dispute if all the checks are positive. Senders will win even if it is unable to provide log information of TTP as in last check. So it is not required the presence of TTP at the time of dispute.

Repudiation of Origin:

If A, B and C denies sending messages m_A, m_B, and m_C respectively, the N can present evidence in the form of treble signature of A, double signature of B and double signature of C plus (t_{id}, em(A), em(B), em(C), dek(A), dek(B), dek(C), md1, md3, md4, md5, K_A, K_B, K_C, m_A, m_B, m_C, eks_from_TTP) to arbitrator. The arbitrator will compare the t_{id} and check

$$em(A) = eK(m_A) \qquad em(B) = eK(m_B) \qquad em(C) = eK(m_C)$$

$$md1 = MD(em(A)) \qquad md3 = MD(em(id\_N)) \qquad md4 = MD(em(B))$$

$$md5 = MD(em(C))$$

$$ek\_from\_A = eP_N(K_A) \qquad dek(A) = eP_{TTP}(ek\_from\_A)$$

$$ek\_from\_C = eP_N(K_C) \qquad dek(C) = eP_{TTP}(ek\_from\_C)$$

$$ek\_from\_B = eP_N(K_B) \qquad dek(B) = eP_{TTP}(ek\_from\_B)$$

A's signature $sS_A(t_{id} \mid md1 \mid md2 \mid md3)$ B's Dual signature C's Dual signature

TTP's signatures $dS_{TTP}(dek(A))$, $dS_{TTP}(dek(B))$ and $dS_{TTP}(dek(C))$

Recipient N will win the dispute if all the checks are positive.

4.7.5 Security Requirements:

Important security requirements are as follow.

Fairness of Protocol:

This protocol has the ability to send messages from involved entities to the recipient N and also among involved entities using the double encrypted key and PKI respectively without the loss of fairness.

There is no violation of fairness if the protocol ends at step1, step 4 and step 5 because of any misbehavior or miscommunication. At this time the intermediate nodes or the recipient N has an encrypted messages and double encrypted keys but they cannot get access to those messages until TTP decrypt the key. If any intermediate node tries to get access to the secret key, he needs to identify himself by decrypting id_message and this is not possible because id_message is encrypted with public key of the recipient N.

The recipient N gets access to the entire original message after step 9. After this step N can misbehave in two ways. Recipient N does not take step 10. TTP detects the misbehavior of the recipient when timeout has been reached and TTP does not receive the recipient N's signature. In this case TTP sends $sS_N(t_{id} | md1 | md3 | md4 | md5)$ and eks_from_TTP to the A. The originator A proves the misbehavior by presenting t_{id}, em(s), id_message and dek(s) and showing that the em(s) and the id_message are matched to $sS_N(t_{id} | md1 | md3 | md4 | md5)$ and dek(s) are matched to eks_from_TTP.

The recipient N can also misbehave by deliberately signing on a fake key in step 10 to reject the transaction later. In this case the originator A can show dishonesty of the recipient by showing that the keys sent at step 1, step 4, and step5 are matched with eks_from_TTP at step 9 but not with the signature2 of the N in step 10.

The protocol also prevents the originator A or intermediate nodes from sending an invalid message or denying sending a message. In the final step, sender can get a receipt in the form of digital signature. However, final step cannot be reached if the originator A or intermediate nodes has sent an invalid message because the recipient N can check the integrity of message by comparing the message digest of encrypted messages and double encrypted keys with the their signatures. If the recipient N did not receive the encrypted message and double encrypted key correctly, it would not give the first signature. The Recipient N would not send the second signature if it cannot access the encrypted messages with the secret keys. In court, the recipient N can present these keys, message digests of keys that he received and signatures of the originators. There is no way that the originator A or intermediate nodes can deny having sent message or double encrypted keys because of their signatures. If they deny from these encrypted messages or double encrypted keys and claim that they sent different encrypted messages and double encrypted keys (which is different from the original encrypted messages and double encrypted keys received by the recipient N), the recipient N can clear his position and disprove their claim by showing their signatures on em(s), dek(s) and id_N that has been received.

Consider the situation when the TTP and the originator team up against the recipient N. On receiving the double encrypted key, TTP does not send the encrypted key to the receiver and claims it did. Now the receiver already gets the signatures of senders so that it can have partial evidence on the double encrypted key. The TTP cannot argue that it gave encrypted key without giving it to the receiver, because the receiver would not give the second signature until it get the encrypted key.

Protection and integrity of Message:

In this protocol there are two types of message, one is only for the recipient N and other is among the involved entities. We used PKI in order to protect the messages among involved entities. Every entity sends the message to the recipient N and wants that no one else can access this message. This type of message is encrypted with the double encrypted key.

The protocol generates a new transaction id (t_{id}) for every new transaction to protect from replay attacks.

Confidentiality of transaction:

The protocol provides the confidentiality so that, no one including the TTP can access the original contents of messages but the recipient N. The recipient N needs to identify itself by sending id_message to TTP. Even the intermediate nodes are involved in the protocol but cannot get access to the messages. The only way to access the messages is through the secret keys that encrypt the messages.

Timeliness:

Protocol achieves timeliness as each involved entity can terminate the protocol at any time at his own judgment while maintaining fairness.

4.7.6 Building the BPEL Processes; Case 2:

In above protocol there are five processes, two for the producer A and C, one for the shipper B, one for the buyer N, and one for the TTP.

Producer Process A:

The producer accepts the order and sends the information to the buyer's preferred shipper B. The process A has two partners process B and Process TTP. The sequence of producer process A is as follow.

Begin sequence

- Invoke the process B and send all information.

- Invoke the process TTP to send the encrypted id_message.
- Receive the signatures from TTP.
- Send signatures to the process B.

End sequence

Shipper Process B:

Sequence of the shipper process B is as follow.

Begin sequence

- Receive the message from the producer process A.
- Invoke the producer process B and send message of the process A to the process C.
- Receive the message from the producer process C.
- Invoke the buyer process TTP to send all messages from the producer process A, all messages from the producer process C and its own message.
- Receive the signature form the process A.

End sequence

Producer process C:

Sequence of the producer process C is as follow:

Begin sequence

- Receive message from the shipper process B.
- Invoke internal process to decrypt the message.
- Reply to the shipper process B with information for the buyer process N.
- Receive signatures of the buyer from the process B.

End sequence

Buyer Process N:

Sequence of the buyer process is as follow.

Begin sequence

- Receive the information from the shipper process.
- Assign the id_message to the container to be use in the invoke activity.
- Invoke the process TTP and send the id_message.
- Receive the acknowledgement from the process TTP.
- Assign double encrypted keys and signature1 to the container.
- Reply to the process TTP and send the double encrypted keys and signature1.
- Receive the encrypted keys from the process TTP.
- Reply the process TTP and send signature2.

End sequence

Process TTP:

The fourth BPEL process place at the TTP and its sequence of activities is as follow:

Begin sequence

- Receive id_message from the producer process A.
- Receive id_message from the buyer process N.
- Invoke internal process to compare id_message for identification.
- Reply with the acknowledgement to the buyer process N.
- Receive double encrypted keys and signature1 form the buyer process N.
- Reply with encrypted key to the buyer process N.
- Receive the signature2 from the buyer process N.
- Reply the producer process A to send the signatures of the buyer process N.

End sequence

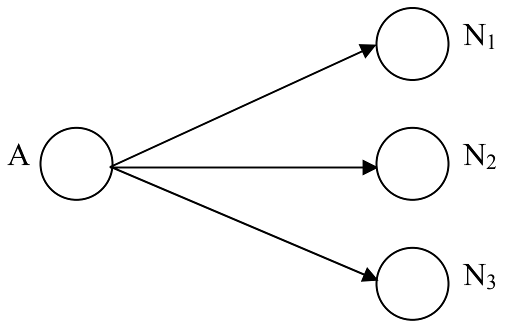
4.8 Non-repudiation Protocol for Multiple Entities:

The protocol presented in pervious sections established non-repudiation in chain linked business transactions. However, in real-time scenarios, messages (same or different) are sent to the multiple entities. There is a chance of communication bottleneck created at the TTP. We propose a multi-entities non-repudiation (MENR) protocol, as an extension of our non-repudiation protocol, such that the sender is able to send different or identical messages to multiple recipients using a single key.

Actual Scenario:

Producer A produces different products and needs to send different messages to multiple receivers. Producer A receives an order from a company to send different products to their different departments N_i. Producer A accepts the order and sends different order fulfillment note including the price, number of units, shipping data of products (size, weight etc) and dates for shipment to the relevant department and does not want anyone else to know this information.

Flow for this multiple entities scenario is as follow.



Issues:

- What factors determine to which recipients should the originator A send the message and the key?

- What factors determine to which recipients should TTP send the encrypted key?
- How does same key open different messages and no one else but the particular recipient can read message?
- How does the originator A know that recipients get the key and hence the original message? etc.

We propose a new non-repudiation protocol in which originator can send different messages to multiple nodes.

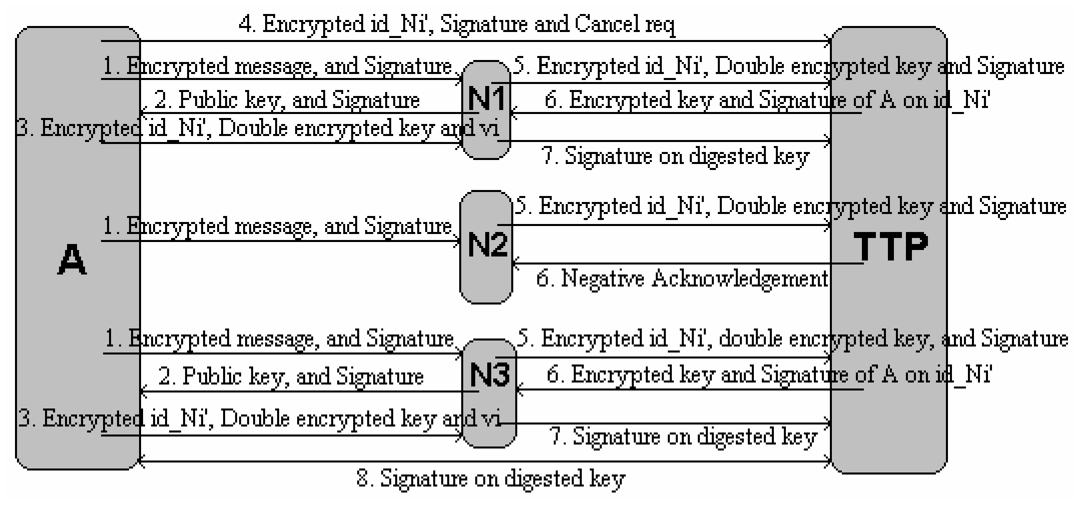
4.8.1 Design and Approach:

An extension by Kremer et al. [24] of a low weight notary protocol for two entities [3] is the first non-repudiation protocol in the literature dealing with multiple entities. This protocol supports a one-to-many topology in which the originator aims to send the same message to multiple recipients. This protocol broadcasts a message among several entities and provides evidence only to those entities who behave honestly during the protocol run, using the same key for encryption. Nevertheless, it is not possible to send different messages to different recipients [23]. We design an optimal protocol named MENR in which originator can send different message to multiple recipients using same key.

Some useful notation in the protocol description is as follows.

A	an originator
N_i	set of intended recipients
$N_i^{'}$	subset of N_i that replied to A with the evidence of receipt
M_i	message being sent from A to a recipient Ni
n_i	random value generated by A

v_i	$eP_{N_i'}(n_i)$: encryption of n_i with N_i''s public key
k	key being selected by A
K_i	k xor n_i: a key for N_i' use to decrypt message
em	$E_{K_i}(Mi)$: encrypted message for N_i' with key K_i
ek_from_A = $eP_{N_i'}(k)$	
dek	= eP_{TTP}(ek_from_A)
ek_from_TTP = dS_{TTP}(dek)	
id_N_i'	message to identify N_i'
md1	MD(em): message digest of encrypted message
md2	MD(dek): message digest of double encrypted key
md3	MD(v_i): message digest of encrypted n_i
md4	MD(id_N_i'): message digest of id_N_i'
N_i''	= $N_i - N_i'$: a subset of N_i with which A wants to cancel the exchange
$Cancel_{req}$	$sS_A(t_{id} \mid TTP \mid N_i'')$: evidence of request of cancellation issued by the originator to the TTP
$Cancel_{Ni''}$	$sS_{TTP}(t_{id} \mid N_i'' \mid Cancel_{req})$: evidence of cancellation issued by the TTP to the Ni''



4. Encrypted id_Ni', Signature and Cancel req
1. Encrypted message, and Signature
2. Public key, and Signature
3. Encrypted id_Ni', Double encrypted key and vi
5. Encrypted id_Ni', Double encrypted key and Signature
6. Encrypted key and Signature of A on id_Ni'
7. Signature on digested key
N1
N2
N3
A
TTP
6. Negative Acknowledgement
5. Encrypted id_Ni', double encrypted key, and Signature
8. Signature on digested key

Figure 4.9: Multiple-Entities Non-repudiation Protocol

Step 1: Originator A broadcasts the encrypted message M_i and his signature on the message digest of the encrypted message to the N_i

$A \longrightarrow N_i : t_{id} \mid A \mid TTP \mid N_i \mid em \mid md1 \mid sS_A(t_{id} \mid md1)$

Step 2: The subset of the recipients wants to receive that message send their public key and signature on their own public key and message digest of the encrypted message.

$N_i' \longrightarrow A : t_{id} \mid A \mid N_i' \mid P_{Ni'} \mid sS_{Ni'}(t_{id} \mid A \mid TTP \mid N_i' \mid P_{Ni'} \mid md1)$

Where $N_i' \in N_i$

Step 3: originator A sends id_N_i' for identification of the N_i', double encrypted key, signature and v_i to the N_i'.

$A \longrightarrow N_i' : t_{id} \mid A \mid TTP \mid N_i' \mid v_i \mid eP_{Ni'}(id\_N_i') \mid dek \mid md2 \mid md3 \mid sS_A(t_{id} \mid md2 \mid md3)$

Step 4: The originator A sends encrypted id_N_i', signature on N_i', and $Cancel_{req}$ to the TTP in order to identify the recipients N_i' and decrypt the key only for correct recipients.

$A \longrightarrow TTP : t_{id} \mid A \mid TTP \mid eP_{TTP}(id\_N_i') \mid sS_A(t_{id} \mid N_i') \mid Cancel_{req}$

Step 5: The recipients N_i' send the double encrypted key, signature and the id_N_i' to the TTP. So the TTP first identify them and then decrypts the key.

$N_i^{'} \longrightarrow TTP : t_{id} \mid A \mid TTP \mid N_i^{'} \mid eP_{TTP}(id\_N_i^{'}) \mid dek \mid md4 \mid sS_{Ni'}(t_{id} \mid md4)$

Step 6: The TTP checks the identification of each the recipient N and do actions as follow.

```
TTP:    FOR (all $N_i^{'} \in N_i$)
        IF ( $N \in N_i^{'}$ )   THEN
            $TTP \longrightarrow N : t_{id} \mid N \mid ek\_from\_TTP \mid sS_A(t_{id} \mid id\_N_i^{'}) \mid sS_{Ni'}(t_{id} \mid md4) \mid$
                              Retrieve signature $sS_N(MD(ek\_from\_TTP))$ from N
        ELSE
            $TTP \longrightarrow N_i^{''} : t_{id} \mid N_i^{''} \mid$ negative Acknowledgement $\mid Cancel_{req} \mid Cancel_{Ni''}$
```

Step 7: The recipients $N_i^{'}$ submit signature to the TTP on a digested secret key.

$N_i^{'} \longrightarrow TTP : t_{id} \mid sS_{Ni'}(MD(ek\_from\_TTP)) \mid sS_{Ni'}(t_{id} \mid md4)$

Step 8: Protocol ends when the originator A fetches signatures of all the recipients ($N_i^{'}$) from the TTP.

$A \longleftrightarrow TTP : t_{id} \mid sS_{Ni'}(MD(ek\_from\_TTP)) \mid sS_{Ni'}(t_{id} \mid md4)$

4.8.2 Dispute Resolution:

As we have mentioned, two kinds of disputes can arise. Now we discuss their resolution.

Repudiation of Recipient:

If N_i denies receiving message 'M_i', A can present evidence in the form of signatures $sS_{Ni'}(t_{id} \mid A \mid TTP \mid N_i^{'} \mid P_{Ni'} \mid md1)$, $sS_{Ni'}(t_{id} \mid md4)$ and $sS_{Ni'}(MD(ek\_from\_TTP))$ plus (t_{id}, em, dek, $id\_N_i^{'}$, v_i, n_i, M_i, k, K_i, md1, md4, P_{TTP}, $P_{Ni'}$, ek_from_TTP) to arbitrator. The arbitrator will compare the t_{id} and check

$$v_i = eP_{Ni'}(n_i)$$

$$K_i = k \text{ xor } n_i$$

$$ek\_from\_A = eP_{Ni'}(k)$$

$$dek = eP_{TTP}(ek\_from\_A)$$

$$em = E_{Ki}(Mi)$$

$$md1 = MD(em)$$

$$md2 = MD(dek)$$

$$md4 = MD(id\_N_i^{'})$$

Message digest of ek_from_TTP

N's signatures $sS_{Ni'}(t_{id} \mid A \mid TTP \mid N_i^{'} \mid P_{Ni'} \mid md1)$, $sS_{Ni'}(t_{id} \mid md4)$ and $sS_{Ni'}(MD(ek\_from\_TTP))$

TTP's signature $dS_{TTP}(dek)$ and his log information to get signature.

Sender A will win the dispute if all the checks are positive. Sender A will win even if he is unable to provide log information of TTP as a last check. So it is not required the presence of TTP at the time of dispute. The arbitrator may further interrogate N_i using $Cancel_{req}$ and $Cancel_{Ni''}$ to check the cancellation list from A.

Repudiation of Origin:

If A denies sending message 'M_i', N_i can present evidence in the form of signatures $sS_A(t_{id} \mid md1)$ and $sS_A(t_{id} \mid md2 \mid md3)$ plus (t_{id}, em, dek, $id\_N_i^{'}$, v_i, n_i, M_i, k, K_i, md1, md2, md3, P_{TTP}, $P_{Ni'}$, ek_from_TTP) to arbitrator. The arbitrator will compare the t_{id} and check

$$v_i = eP_{Ni'}(n_i)$$

$$K_i = k \text{ xor } n_i$$

$$ek\_from\_A = eP_{Ni'}(k)$$

$dek = eP_{TTP}(ek\_from\_A)$

$em = E_{Ki}(Mi)$

$md1= MD(em)$

$md2= MD(dek)$

$md3=MD(v_i)$

Message digest of ek_from_TTP

A's signatures sS_A (t_{id} | md1) and $sS_{Ni'}$ (t_{id} | md2 | md3)

TTP's signature $dS_{TTP}(dek)$

Recipient N will win the dispute if all the checks are positive.

4.8.3 Security Requirements:

Important requirements of non-repudiation services are as follow.

Fairness of Protocol:

This protocol has the ability to send different messages to different entities using the same key for encryption without the loss of fairness.

There is no breach of fairness if the protocol ends at step1 because of any misbehavior or miscommunication. The originator already sent the encrypted message $em = E_{Ki}(Mi)$ but still holding the key 'k' and 'n_i' which is use to decrypt that message. If any recipient other than accepted recipients tries to get access the secret key, he needs to identify himself by decrypting id_N_i' and this is not possible because id_N_i' is encrypted with public key of the recipient N_i'. The recipient N_i' gets access to entire original message after step 6. After this step N_i' can misbehave in two ways. Recipient N_i' does not take step 7. TTP detects the misbehavior of receiver when timeout has been reached and TTP does not receive recipient's signature. In this case TTP sends $sS_{Ni'}$ (t_{id} | md4) and ek_from_TTP to the originator A. Originator A proves the misbehavior by presenting t_{id}, em, id_ $N_i^{'}$, $P_{Ni'}$, n_i and dek and shows that 'em' is matched to the $sS_{Ni'}$ (t_{id} | A | TTP | $N_i^{'}$ | $P_{Ni'}$ | md1) and id_ $N_i^{'}$ is matched to $sS_{Ni'}$ (t_{id} | md4) and dek is matched to ek_from_TTP.

Recipient $N_i^{'}$ can also misbehave by deliberately signing on a fake key in step 7 to refuse the transaction later. In this case the originator A can proves dishonesty of receiver by showing that key sent at step 1 matched with ek_from_TTP at step 6 but not with the signature of the $N_i^{'}$ in step 7.

The protocol also prevents the originator A from sending an invalid message or denying sending a message. In the final step originator can get a receipt in the form of digital signature. However, final step cannot be reached if the originator A has sent an invalid message because the receiver $N_i^{'}$ can check the integrity of message by comparing the message digest of encrypted message after step 1. If $N_i^{'}$ did not receive the encrypted message correctly, he would not give the signature and public key. Recipient $N_i^{'}$ can check the integrity of key by comparing the message digest of double encrypted key after step 3 and if there is some problem, he would not execute next step. Recipient $N_i^{'}$ would not send the signature on digested key if he cannot access the encrypted message with the secret key. In court, the recipient N can presents that key and message digest of the key that he received in step 3 and show that key correspond to its message digest but unable to unlock the message. There is no way the originator A can deny having sent message and double encrypted key because of his signature. If originator A denies and claims that he sent different message and double encrypted key (which is different from original encrypted message and double encrypted message received by $N_i^{'}$) to the recipient $N_i^{'}$, the $N_i^{'}$ can clear his position and disprove that claim by showing A's signatures sS_A (t_{id} |

md1) on the ‘em’ and $sS_{Ni'}$ (t_{id} | md2 | md3) on the dek and the id_ $N_i^{'}$, that has been received.

Now consider the case where the TTP and the originator team up against the recipient $N_i^{'}$. If the TTP does not send the encrypted key to the receiver in step 6 and claims that he did. Now the recipient already got $sS_{Ni'}$ (t_{id} | md2 | md3) so that he can have an evidence on the double encrypted key. TTP cannot argue that he gave the encrypted key without really giving it to the recipient, because recipient would not give $sS_{Ni'}$(MD(ek_from_TTP)) until he gets the encrypted key.

If the recipient $N_i^{''}$ claims that he sent the public key and an acceptance to receive the message and did not get any response from the originator. Originator A can disprove it by showing the list of $N_i^{''}$, $Cancel_{req}$ and TTP’s signature $Cancel_{Ni''}$ which show that the subset of N_i with which the exchange has been cancelled.

Verifiability of Third Party:

If the TTP misbehaves, resulting in the loss of fairness for an entity, the victim can show the reality in a dispute [23].

Confidentiality of transaction:

The protocol provides the confidentiality so that, except the particular recipient no one else including the TTP can access the original contents of message. Even the $N_i^{''}$ involved in the protocol and get a chance to receive the encrypted message but he is unable to open it because he needs the key k and n_i to decrypt that message. If the $N_i^{''}$ gets the key k from some other recipient but he cannot access the message because he needs n_i to compute K_i.

Protection and integrity of Message:

Protection and integrity of the message is provided in same way as in pervious protocols. Further protection is provided by generating different 'n_i' and hence K_i for particular recipient N_i so that no one else can access the key K_i and hence the message even if he knows the key k.

4.8.4 Efficiency:

We compare our protocol with the one where an n-instance of a two-party protocol [20] is used in order to send different messages to the intended parties. For this comparison, we use the following operations:

- encryption and decryption
- signature generation and verification
- modular equation computation
- random numbers generation
- store and fetch actions

Depending on which algorithm is used for each of these operations, the bit complexity of each of the participants will change but the relation going between them remains.

We denote [23]:

$\approx$ roughly equal

$>$ or $<$ greater or smaller

$>>$ or $<<$ much greater or much smaller

Table 4.1

TTP's computation complexity

n-instanced two parties	Comparison	Our Approach
Decryption of n of keys	>>	Decryption of one key

Hence we can see in Table 1 that efficiency of the TTP is improved when it is generalized to multiple entities.

Table 4.2

A's computation complexity

n-instanced two parties	Comparison	Our Approach
Generating n keys	>>	Generating one key
Double encrypting n keys	>>	Encrypting one key
No generation of n_i and encrypting it	<<	Generation of n_i and encrypting it
N dual signatures	≈	n signatures on message digest of em, dek and v_i
No identification required	<<	Generation of id_ N_i' and encrypt it with $P_{Ni'}$
n fetches operation to get sS_N(MD(ek_from_TTP))	>>	One fetch operation to get $sS_{Ni'}$(MD(ek_from_TTP))

Above table shows that A's efficiency is improved. In n-instances of two parties protocol, the misbehavior of the recipients disclose when they already get the key and hence the message. In this case all computations of the TTP get wasted. In our protocol, the N can get the key only after identification which reduces the loss.

Computation of $N_i^{'}$ slightly increased because of identification and n_i, however if the recipient and the originator have pervious strong relational and they may share the secret, then the identification and encryption of n_i could be avoided in each protocol run though it should be still included in evidence.

Table 4.3

N's computation complexity

n-instanced two parties	Comparison	Our Approach
Signature 1	=	Signature $sS_{Ni'}(t_{id}$ | A | TTP | Ni' | $P_{Ni'}$ | md1)
Signature 2	=	Signature $sS_{Ni'}$(MD(ek_from_TTP)
No signature on id_Ni'	<<	Signature on id_Ni'
Obtained encrypted key from TTP and decrypt it	<	Obtain encrypted key from TTP and decrypted it to get k plus decrypt n_i and compute K_i

Our protocol is extremely efficient such that it exchange different messages among multiple recipients using only one key for evidence distribution and reducing the computation for the originator and the TTP. We get this new feature with slight increase in cost of public key encryption and decryption of n_i and id_$N_i^{'}$. This protocol is efficient than any other two-party protocols, since it allow to send different messages to multiple entities in a confidential way as well as to cancel the protocol for a group of recipient N_i''. In addition it provides timeliness, as each entity can terminate the protocol at any time while maintaining the fairness.

4.8.5 Building BPEL processes for the MENR Protocol:

In this protocol number of processes depend on the number of recipients participated in protocol. At least there are three processes with one recipient process, originator process and TTP process.

Originator Process A:

Originator process starts when he broadcasts the encrypted message M_i and his signature on the message digest of the encrypted message to the recipient processes. Sequence of originator process is as follow.

Begin sequence

- Invoke recipient processes to send encrypted message and signature.
- Use pick construct that allows waiting for suitable message to arrive or for a time out alarm to go off so that originator can receive message from the recipients and if he does not receive anything from any recipient with in that time out alarm then originator process able to send cancel request to the TTP process.
- Send encrypted id_N_i', double encrypted key, signature and v_i to the recipient processes N_i' through the same channel of invoke activity.
- Invoke the TTP process to send encrypted id_N_i', signature on N_i', and $Cancel_{req}$.
- Receive signature from the TTP process.

End sequence

Recipient Process N:

Sequence of the recipient process is as follow.

Begin sequence

- Receive encrypted message and signature from the originator process.
- Reply to the originator process with his public key and signature.
- Receive encrypted $id\_N_i^{'}$, double encrypted key, signature and v_i from the originator process.
- Invoke the TTP process to send encrypted $id\_N_i^{'}$, double encrypted key, and signature.
- Receive encrypted key from the TTP process.
- Reply the TTP process with his signature on digested key.

End sequence

TTP Process:

Sequence of the TTP process is as follow.

Begin sequence

- Receive information from the originator process as shown in figure 4.7.
- Receive encrypted $id\_N_i^{'}$, double encrypted key, and signature from the recipient processes.
- Reply to the recipient processes with the encrypted key if there $id\_N_i^{'}$ matches otherwise send $Cancel_{req}$. So switch activity can be use to perform this step.
- Use pick construct so that if he does not receive anything from any of the recipients with in that time out alarm then TTP process able to show misbehavior of the recipient process(es).
- Reply to the originator process with the signatures of recipients.

End sequence

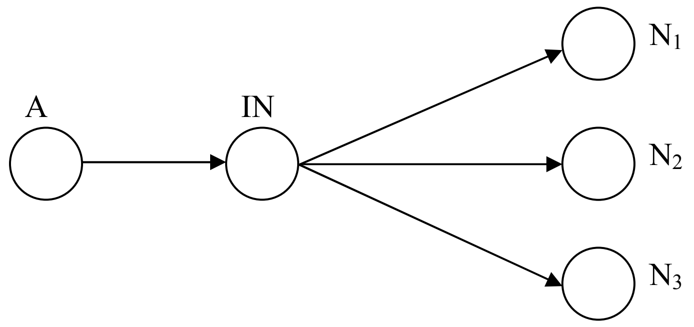
4.9 Non-repudiation Protocol for chain-linked Multiple Entities:

The protocol presented in pervious sections established non-repudiation in chain linked business transactions in which the originator can send a message to the recipient involving some intermediate nodes. In some cases the originator needs to send different message to the recipients through the intermediate node. We propose a chain-linked multi-entities non-repudiation (CLMENR) protocol, is the extension of our non-repudiation protocols, such that the sender is able to send different to the multiple recipients through intermediate nodes.

Actual Scenario:

Scenario is same as in chain linked business transaction but supplier A wants to send different messages to multiple recipients and does not want anyone else to know this information.

Flow for this chain-linked multiple entities scenario is as follow.



4.9.1 Design and Approach:

We design an optimal protocol named CLMENR in which originator can send different message to multiple recipients and we are introducing intermediate node IN which does play the role of a hub and reducing load on the originator.

Most of the notations are same as in MENR except following.

Dual signature = t_{id} | md1 | md2 | $sS_A(t_{id}$ | md1| md2)

md1	MD(em): message digest of encrypted message
md2	MD(dek): message digest of double encrypted key
md3	MD(v_i): message digest of encrypted n_i
md4	MD(id_$N_i^{'}$): message digest of id_$N_i^{'}$

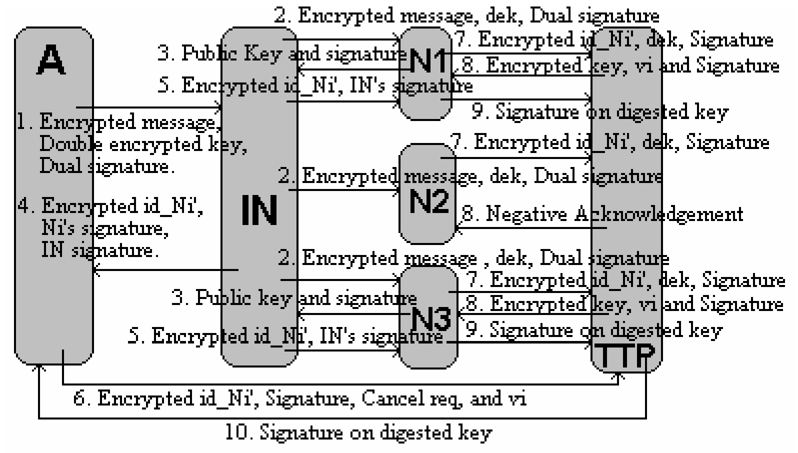

A
IN
N1
N2
N3
TTP
1. Encrypted message, Double encrypted key, Dual signature.
2. Encrypted message, dek, Dual signature
3. Public Key and signature
4. Encrypted id_Ni', Ni's signature, IN signature.
5. Encrypted id_Ni', IN's signature
6. Encrypted id_Ni', Signature, Cancel req, and vi
7. Encrypted id_Ni', dek, Signature
8. Encrypted key, vi and Signature
8. Negative Acknowledgement
9. Signature on digested key
10. Signature on digested key

Figure 4.10: Chain-linked multiple entities Non-repudiation protocol

Step 1: The Originator A send the encrypted message M_i, double encrypted key and dual signature to the IN.

A $\longrightarrow$ IN : t_{id} | IN | A | TTP | N_i | em | dek | Dual signature

Step 2: The intermediate node IN broadcasts all this information to the N_i.

IN $\longrightarrow$ N_i : t_{id} | IN | A | TTP | N_i | em | dek | Dual signature

Step 3: The subset of recipients requests to receive that message send their public key and signature on their own public key and message digest of the encrypted message.

$N_i^{'}$ $\longrightarrow$ IN : t_{id} | A | TTP| $N_i^{'}$ | IN | $P_{Ni'}$ | $sS_{Ni'}$ (t_{id} | A | TTP | $N_i^{'}$ | $P_{Ni'}$ | md1)

Step 4: The intermediate node IN forward these signatures plus encrypted id_$N_i^{'}$ and his signature on md4 to the originator A.

$IN \longrightarrow A : t_{id} \mid A \mid N_i' \mid \mid eP_A(id\_N_i') \mid md4 \mid sS_{IN}(t_{id} \mid md4) \mid sS_{Ni'}(t_{id} \mid A \mid TTP \mid N_i' \mid P_{Ni'} \mid md1)$

Step 5: The intermediate node IN sends encrypted id_N_i' and his signature on md4 to the recipients N_i'.

$IN \longrightarrow N_i' : t_{id} \mid N_i' \mid eP_{Ni'}(id\_N_i') \mid md4 \mid sS_A(t_{id} \mid md4)$

Step 6: The originator A sends v_i, encrypted id_N_i' , his signature on md3 and md4 and $Cancel_{req}$ to the TTP in order to identify N_i' and decrypt the key only for correct recipients.

$A \longrightarrow TTP : t_{id} \mid A \mid TTP \mid N_i' \mid v_i \mid md3 \mid md4 \mid eP_{TTP}(id\_N_i') \mid sS_A(t_{id} \mid md3 \mid md4) \mid Cancel_{req}$

Step7: The recipients N_i' send the double encrypted key, signature and the id_N_i' to the TTP. So the TTP first identify them and then decrypts the key.

$N_i' \longrightarrow TTP : t_{id} \mid A \mid TTP \mid N_i' \mid eP_{TTP}(id\_N_i') \mid dek \mid md4 \mid sS_{Ni'}(t_{id} \mid md4)$

Step 8: The TTP checks the identification of each the recipient N and do actions as follow.

```
TTP:   FOR (all $N_i' \in N_i$)
          IF ( $N \in N_i'$ )   THEN
               $TTP \longrightarrow N : t_{id} \mid N \mid ek\_from\_TTP \mid v_i \mid md3 \mid sS_A(t_{id} \mid md3 \mid md4) \mid$
                         Retrieve signature $sS_N(MD(ek\_from\_TTP))$ from N
          ELSE
               $TTP \longrightarrow N_i'' : t_{id} \mid N_i'' \mid$ negative Acknowledgement $\mid Cancel_{req} \mid Cancel_{Ni''}$
```

Step 9: The recipients N_i' submit signature to the TTP on a digested secret key.

$N_i' \longrightarrow TTP : t_{id} \mid A \mid TTP \mid sS_{Ni'}(MD(ek\_from\_TTP)) \mid sS_{Ni'}(t_{id} \mid md4)$

Step 10: Protocol ends when the originator A fetches signatures of all the recipients ($N_i^{'}$) from the TTP.

A $\leftrightarrow$ TTP : t_{id} | $sS_{Ni'}$(MD(ek_from_TTP)) | $sS_{Ni'}$ (t_{id} | md4)

4.9.2 Building BPEL processes for the CLMENR Protocol:

In CLMENR protocol like MENR protocol, number of processes depend on the number of recipients participated in protocol. At least there are four processes with one recipient process, originator process, intermediate process and TTP process.

Originator Process A:

Sequence of originator process is as follow.

Begin sequence

- Invoke intermediate process to send encrypted message, double encrypted key and dual signature.
- Receive encrypted id_$N_i^{'}$, $N_i^{'}$s signatures and IN's signature from the intermediate process.
- Invoke the TTP process to send encrypted id_$N_i^{'}$, $cancel_{req}$ and signature.
- Receive signatures of recipient from the TTP process.

End sequence

Intermediate Process IN:

Sequence of the intermediate process is as follow.

Begin sequence

- Invoke recipient processes to send encrypted message double encrypted key and dual signature.

- Use pick so that intermediate process can receive message from the recipients and if he does not receive anything from some of the recipients with in that time out alarm then intermediate process able to send cancel request from those recipients to the originator process.
- Send encrypted id_N_i', and IN's signature to the recipient processes N_i' through the same channel of invoke activity.
- Reply the originator process in order to send encrypted id_N_i', signature on N_i', and his own signature.

End sequence

Recipient Process N:

Sequence of the recipient process is as follow.

Begin sequence

- Receive encrypted message, double encrypted key and dual signature from the intermediate process.
- Reply to the intermediate process with his public key and signature.
- Receive encrypted id_N_i' and IN's signature from the intermediate process.
- Invoke the TTP process to send encrypted id_N_i', double encrypted key, and signature.
- Receive encrypted key from the TTP process.
- Reply the TTP process with his signature on digested key.

End sequence

TTP Process:

Sequence of the TTP process is as follow.

Begin sequence

- Receive information from the originator process as show in figure 4.8.
- Receive encrypted $id\_N_i^{'}$, double encrypted key, and signature from the recipient processes.
- Reply to the recipient processes with the encrypted key if there $id\_N_i^{'}$ matches otherwise send $cancel_{req}$. So switch activity can be use to perform this step.
- Use pick construct so that if he does not receive anything from any of the recipients with in that time out alarm then TTP process able to show misbehavior of the recipient process(es).
- Reply to the originator process with the signatures of recipients.

End sequence

Chapter 5

Conclusion

In this era of globalization Web Services are considered as a future of internet. BPEL is a way of integrating those web services in order to get the best, simple, and economical service in B2C as well as in B2B transactions. With the increase of World Wide Web usage it is necessary that there are some efficient and fair approaches of security. Non repudiation is special technique in which the scope of the system is particularly wide, as it includes agents outside the communication exchange. Getting a protocol right involves taking account of many great possible loopholes. There are several publications that address Non-repudiation dilemma using various levels of trust and dependency on a third party and with different weaknesses.

The Non-repudiation protocols developed and specified in BPEL are based on a protocol which possesses several new challenges in different scenarios. In our approach, the trusted third party signature is not considered as evidence therefore TTP availability is not required at the time of dispute resolution. We extended this protocol to some real time scenarios. In chained linked transactions no one else but the recipient accesses the message because he needs to identify himself before proceeding. One of our proposed protocols has a uniqueness in which intermediate nodes can access message and modify it according to the requirements. We also proposed a protocol that has an ability to send different messages to several recipients using a single key that reduces the load of generating the key by the originator and decrypting that key by the TTP. We also enhanced this protocol and reduced the communication load on the originator by introducing an intermediate node that is responsible to interact with the recipients. Protocols are analyzed so that they fulfill the security and non-repudiation requirements in efficient manner. We used Petri nets to validate the flow of protocols.

In the multiple entities non-repudiation protocols the number of recipients may vary, so as the number increases there is a chance that the entities are unstable under the load and system may crash resulting in the productivity loss, user frustration, delays, system outage, and data loss/corruption.

Future works include situations where two or more entities can team-up to cause problems for other entities. Since the role of TTP is very important, there should be a protocol that involves multiple trusted third parties for economical TTP(s) availability. Since the protocols are based on the assumption of reliable communication channel, protocol independent of reliable communication channels is needed.

References

[1] J. Zhou, "*Non-repudiation in Electronic Commerce*," Artech House, Computer Security Series, 2001.

[2] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. "*Business Process Execution Language for Web Services*", May 2003 Published on the World Wide Web by BEA Corp., IBM Corp., Microsoft Corp, SAP AG and Siebel Systems, Version 1.1, Pages 8-111, 2003.

http://www-128.ibm.com/developerworks/library/ws-bpel/

[3] J. Zhou and D. Gollmann. "*A Fair Non-repudiation protocol*". In Proceeding of 1996 IEEE Symposium on Security and Privacy (S & P '96), Oakland, CA, Pages 55-61, IEEE Computer Society Press, Los Alamitos, CA, 1996.

[4] K. Kim, S. Park, and J. Baek, "*Improving fairness and privacy of Zhou-Gollmann Fair Non-repudiation Protocol*," In Proceeding of 1999 ICPP Workshops on Security (IWSEC), Pages 140- 145 IEEE computer Society, 1999.

[5] S. Masud, "*Building Reliable Asynchronous Processes with BPEL4WS*" IBM Developer Works, Use Rosetta Net-based Web Services, Part 4: BPEL4WS and Rosetta Net, 2003.

www-106.ibm.com/developerworks/ webservices/library/ws-rose4/

[6] C. Peltz. ”A *look at WSCI and BPEL4W,*”In Web services Journal, WSJ feature, 2004.

http://www.sys-con.com/webservices/articleprint.cfm?id=592

[7] R. Khalaf, N. Mukhi, and S. Weerawarana. “*Service-Oriented Composition in BPEL4WS”* IBM T.J Watson Research Center, NY, 2003

http://www.www2003.org/cdrom/papers/alternate/P768/choreo_html/p768-khalaf.htm

[8] F. Curbera, M. Duftler, R. Khalaf, N. Mukhi, W. Nagy, and S. Weerawarana. ”*Business Process Execution Language for Web Services Java Runtime (BPWS4J)”*, IBM T.J Watson Research Center, NY, 2002.

[9] N. Mukhi, R. Khalaf, W. Nagy, M. Duftler , F. Curbera, S. Weerawarana et.al. *“Business Process with BPEL4WS: All Columns”* (developerWorks) offers additional insight on writing business processes with BPEL4WS.

http://www-128.ibm.com/developerworks/webservices/library/ws-bpelcol.html

[10] T. Murata, “*Petri Nets: Properties, Analysis and Applications”*, Proceedings of the IEEE, Vol. 77, NO. 4, Pages 541-580 April 1989.

[11] J.P. Thomas, N. Nissanke, and K.D. Baker, “*A Hierarchical Petri Net Framework for the Representation and Analysis of Assembly*”, IEEE Transactions on Robotic and Automation, Vol. 12, NO. 2, Pages 268-279, April 1996.

[12] A. J. Muhammad. *“Petri Net Modeling of Web Services”*, Master Degree Thesis at Computer Science department, Oklahoma State University, Ch. 4, April 2003.

[13] J. Zhou and D. Gollmann. Evidence and Non-repudiation “*Journal of Network and computer Applications”* 20(3): Pages 267-281, July 1997

[14] B. Crispo, S. Etalle, and W. J. Fokkink, "*Accountability in Electronic Commerce Protocols*". In Preceding Research Proposal Computer Science open Competition 2003.

[15] S. Yang, Stanley Y. W. Su, and H. Lam, "*A Non-Repudiation Message Transfer Protocol for E-commerce*". Proceeding of the IEEE International Conference on E-commerce (CEO'03), IEEE Computer Society, 2003.

[16] H. Adams. "*Asynchronous Operations and Web Services part 3*", Add business semantics to Web services Oct, IBM T.J Watson Research Center, NY, 2002. www-106.ibm.com/developerworks/webservices/library/ws-asynch3

[17] B. Atkinson, G. Della-libera, S. Hada, P. Hallam-Baker, J. Klein, B. LaMachia, P. Leach, J. Manferdeelli, H. Maruyama, A. Nadalin, N. Nagaratnam, H. Prafullchandra, J. Shewchuk, D. Simon, "*Web Services Security(WS-security)*" version 1.0 Developer works by IBM, Microsoft, and VeriSign, April 2002. http://www-106.ibm.com/developerworks/webservices/library/ws-secure/

[18] K. Salimifard, M. Wright*, "Petri net-based modeling of workflow systems: An overview"*, Management Science Department, Lancaster University, Bailrigg, Lancaster LA1 4YX, UK. European Journal of Operational Research 134, Pages 664-676, 2001.

[19] K. Jensen, *"Colored Petri Nets: Basic Concepts, Analysis Method and Practical Use"*, Vol. 1: Basic concepts, EATCS Monograph on Theoretical Computer Sciences, Springer, Berlin, 1992.

[20] V. Atluri, W. Huang, "*A Petri net Based Safety Analysis of Workflow Authorization"*, MSIS Department and Center for Information Management, Integration and Connectivity, Rutgers University, Newark. Journal of Computer Security 8, Pages 209-240, 2000.

[21] V. Atluri, W. Huang, "An Authorization Model of Work Flows", In proceeding of the fifth European Symposium on research in Computer Science Security, Pages 44-64, 1996.

[22] S. Hong, K. Kim. *"A Reliability Analysis of Distributed Programs With Colored Petri Nets"* ISDN Call Processing Section. ETRI 16 1 Kajong-Dong,Yusong-Gu,Taejon 305-350, Department of Computer and Informations, Seokyeong University, Seoul Korea, IEEE, Pages 3975-3980, 1997.

[23] J. Onieva, J. Zhou, J. Lopez. "*Non-repudiation protocols for multiple entities"* Computer Science Department, E.T.S. Ingenieria Informatica University of Malaga, Spain, Institute for Infocomm Research, Singapore, Computer Communications 27, Pages 1608–1616, 2004.

http://www.elsevier.com/wps/find/journaldescription.cws_home/525440/description#description

[24] S. Kremer, O. Markowitch, "*A multi-party non-repudiation protocol*", In: Proceedings of 15th IFIP International Information Security Conference, Kluwer Academic Publishers, Dordrecht, Pages 271–280, 2000.

[25] J. Onieva, J. Zhou, M. Carbonell, and J. Lopez, "*A multiparty non-repudiation protocol for exchange of different messages*". In Proceedings of 18th IFIP International Information Security Conference, Athens, Greece, Pages 37-48, May 2003.

[26] J. Onieva, J. Zhou, M. Carbonell, J. Lopez, "*Intermediary Non-repudiation Protocols*", In Proceedings of IEEE Conference on Electronic Commerce, IEEE Computer Society Press, Silver Spring MD, Pages 207–214, 2003.

[27] "*BPWS4J*" Developers Work, IBM, Version 2.1, 2002

http://www.alphaworks.ibm.com/tech/bpws4j

GLOSSARY

Glossary A

BPEL and Web services Terminologies

Binding: Describes the protocol being used to carry the Web Service communication; bindings currently exist for SOAP, HTTP GET, HTTP POST, and MIME.

BPEL4WS: Business process execution language for web services is a language for the formal specification of business processes and business interaction protocols.

Service Link Type: The service link types of the WSDL document represent the interaction between the order service and each of the parties with which it interacts.

SOAP: Simple Object Access Protocol is a platform-independent protocol that uses XML to make remote procedure call over HTTP. Each call and response is packaged in a SOAP message—an XML message contain all the information necessary to process its contents.

Web Services: A web services is a class store on the machine that can be accessed on another machine over a network.

WSDL: Web Services Description language, an XML document that describes how a web services behave. A WSDL defines the methods that the web services makes available and the way in which client can interact with those methods.

WS-Security: Web services security language or WS-Security is designed to be used as the basis for the construction of a wide variety of security models including PKI, Kerberos, and SSL. Specifically WS-Security provides support for multiple security tokens, multiple trust domains, multiple signature formats, and multiple encryption technologies.

BPEL Activities:

The executable actions take place with in BPEL process is called activities. There are following BPEL activities [2].

Receive: The <receive> construct allows the business process to do a blocking wait for a matching message to arrive.

Reply: The <reply> construct allows the business process to send a message in reply to a message that was received through a <receive>. The combination of a <receive> and a <reply> forms a request-response operation on the WSDL portType for the process.

Invoke: The <invoke> construct allows the business process to invoke a one-way or request-response operation on a portType offered by a partner.

Assign: The <assign> construct can be used to update the values of variables with new data. An <assign> construct can contain any number of elementary assignments

Throw: The <throw> construct generates a fault from inside the business process.

Wait: The <wait> construct allows you to wait for a given time period or until a certain time has passed. Exactly one of the expiration criteria must be specified.

Empty: The <empty> construct allows you to insert a "no-op" instruction into a business process.

Sequence: The <sequence> construct allows you to define a collection of activities to be performed sequentially in lexical order.

Switch: The <switch> construct allows you to select exactly one branch of activity from a set of choices.

While: The <while> construct allows you to indicate that an activity is to be repeated until a certain success criteria has been met.

Pick: The <pick> construct allows you to block and wait for a suitable message to arrive or for a time-out alarm to go off. When one of these triggers occurs, the associated activity is performed and the pick completes.

Flow: The <flow> construct allows you to specify one or more activities to be performed concurrently. Links can be used within concurrent activities to define arbitrary control structures.

Scope: The <scope> construct allows you to define a nested activity with its own associated variables, fault handlers, and compensation handler.

Compensate: The <compensate> construct is used to invoke compensation on an inner scope that has already completed normally. This construct can be invoked only from within a fault handler or another compensation handler.

Glossary B

Non-repudiation Terminologies

Non-repudiation: Non-repudiation ensures that the originator of a message cannot deny having sent the message or receive of a message cannot deny having received the message.

Double encrypted Key: A twice-encrypted secret key that is first encrypted with the receiver public key and then with the public key of TTP.

Non-repudiation of origin (NRO): is considered to protect the recipient from the originator falsely denying having sent the message.

Non-repudiation of Receipt (NRR): is considered to protect the originator from the recipient falsely denying having received the message.

Non-repudiation of Delivery (NRS): is provided the originator with the evidence that message has been submitted for delivery to the recipient.

Non-repudiation of Delivery (NRD): is provided the originator with the evidence that message has been sent from delivery agent to the recipient.

Secret Key: symmetric = receiver or transmitter share secret key, nobody else. It is randomly generated at runtime.

Glossary C

Petri Net Terminologies

$m(p)$: it is used to denote distinct color e.g. $m(p) = g + r$ represents place p containing a token of color g and a token of color r, i.e., $CR(m(p)) = \{g, r\}$.

$CR(p)_{MS}$: Represents the set of multi set or bags over CR(p) e.g. given a set $CR(p) = \{a, b, \ldots.\}$, the multi sets a, a+b, a+2b are members of CR(p).

VITA

Muhammad Bilal

Candidate of the Degree of

Master of Science

Thesis: ‘FAIR’ BPEL PROCESSES TRANSACTION USING NON-REPUDIATION PROTOCOLS

Major Field: Computer Science

Biographical:

Personal Data: Born in Lahore, Pakistan, on August 23, 1974, son of Mr. and Mrs. Muhammad Mukhtar

Education: Received the Bachelor of Mechanical Engineering Degree from University of Engineering and Technology, Lahore, Pakistan in May 2000. Completed the requirements for the Master of Science Degree in Computer Science at the Computer Science Department at Oklahoma State University in May, 2004.

Experience: Employed by Lah-soft Pak, Lahore, Pakistan, as Web Developer, January 1999 to July 1999; employed by Office Automation Services, Karachi, Pakistan, as Software Engineer, January 2000 to July 2000: employed by Accenture, NY, USA as a QA Software Analyst, March 2004 to September 2004.

Name: Muhammad Bilal Date of Degree: May, 2005

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: 'FAIR' BPEL PROCESSES TRANSACTION USING NON-REPUDIATION PROTOCOLS

Pages in Study: 98 Candidate for the Degree of Master of Science

Major Field: Computer Science

Scope and Method of Study: The single most important invention that has completely revolutionized how business transactions are conducted is the internet. The fast paced technological advancement of web services standards and its tools have transformed the world wide web from information sharing platform to an extremely powerful and open ecosystem of e-services that not only delivers the information but also provide decision support, transactions and applications. There is a need for powerful protocols to achieve universal interoperability among web services and to provide a fair and secure and accountable environment. BPEL provides a language for the formal specification of business processes and business interaction protocols. In business transactions Non-repudiation is a serious and troublesome security issue in which any involved party denies having participated in a transaction. In this thesis – we propose and verify novel non-repudiation protocol specification in BPEL.

Findings and Conclusion: We model non-repudiation protocols in BPEL and analyze those using Petri Nets. We also propose new Non-repudiation protocols for chain-linked business transactions. In a business transaction there may be more then one recipient and different messages to each of them. We therefore also propose protocols for multiple recipients. We show that the proposed protocols meet the security requirements and are terminated when anyone of the transactions fails, without losing fairness. Our proposed protocols fulfill the requirements of security, fairness, protection and timeliness in different scenarios. Computation load of originator and trusted third party are also reduced using these approaches. These protocols are modeled as Color Petri Nets to verify the reliability of the protocols. BPEL processes have been specified using these protocols.

ADVISOR'S APPROVAL: Johnson Thomas