Model Checking for E- Commerce Transaction

Fraisy Varghese
Software Developer Trainee
Izone Info Soft
Irinjalakuda, Thrissur.

Abstract- **E-commerce is a modern business methodology that address the needs of organizations, merchants, and consumers to cut costs while improving quality of goods and services and increasing the speed of service delivery. The fast development of e-commerce has necessitate the development of e-commerce protocols. These protocols guarantee the confidentiality and integrity of information exchanged. In addition, researchers have identified other desirable properties, such as, money atomicity, goods atomicity and validated receipt that must be satisfied by e-commerce protocols. This seminar provides a brief introduction on how model checking can be used to obtain an assurance about the existence of these properties in an e-commerce protocol. It is important that these desirable properties be satisfied, even in the presence of site or communication failure. Using the model checker we evaluate which failures cause the violation of one or more of the properties. The results of the analysis are then used to propose a mechanism that handles the failures to make the protocol failure resilient**.

Index Terms: Modern checking, securing e-commerce, securing transaction

I INTRODUCTION

E-commerce can be defined as a modern business methodology that address the desire firms, consumers, and management to cut costs while improving the quality of goods and increasing the speed of services. E-commerce is termed as a new online approach to performing traditional functions such as payment and funds transfer ,order entry and processing ,invoicing, inventory management, cargo tracking, electronic catalogs, and point-of-sales data gathering .Number of e-commerce protocols developed due to the popularity of e-commerce. Most of these protocols ensure that the information exchanged between the parties is protected from unauthorized revelation and modification. Moreover, researchers have recognized several other desirable properties of e-commerce protocols such as money atomicity and goods atomicity and validated receipt.[1]Money atomicity ensures that money is neither created nor destroyed in the course of an e-commerce transaction. Goods atomicity ensures that a merchant receives payment if and only if the customer receives the product. Validated receipt ensures that the customer is able to verify the contents of the product about to be received, before making the payment. Although such properties have been identified, a major problem is verifying if a given e-commerce protocol satisfies these properties, especially in the presence of network and site failures. Here we concentrate on the problem of protocol verification using an existing software verification technique known as model checking. Model checking, an approach based on exhaustive search of finite state spaces, could be applied to this system to verify its properties.[4] A model of this system and a property specification could be given as input to a model checker, which would return a yes, meaning that the properties were verified, or provide a counter example. The reasons for using model checking are as follows. 1) Model checking is a completely automated technique and considerably faster than other approaches, such as, manual proofs and simulations. 2) if a property does not hold, a counter example is produced by the model checker which helps in understanding why the property does not hold. 3) Model checking has previously been used successfully to verify security protocols.

II.PROTOCOL FUNDAMENTALS

Fig. 1 represents the high-level abstraction of the protocol, and its processes are summarized as follows: Messages are exchanged between a customer, a merchant and a trusted third party (TTP) [2]. A merchant has several products to sell. The merchant places a description of each product on an online catalog service with a TTP, along with a copy of the encrypted product. When a customer finds a product of interest by browsing the catalog, he or she downloads the encrypted product and then sends a purchase order to the merchant. The customer cannot use the product unless it has been decrypted, and the merchant does not send the decrypting key unless the merchant receives a payment token through the purchase order process. The customer, in turn, does not pay unless he or she is sure that the correct and complete product has been received. The TTP provides anonymous support for purchase order validation, payment token approval, and approval of the overall transaction between the customer and the merchant. Given these assumptions, the detailed steps of the protocol are as follows. (A use case diagram of these processes is depicted in Fig. 2; the corresponding sequence diagram is shown in Fig. 3.)First, the customer browses the product catalog
Located at the TTP and chooses a product. The customer then downloads the encrypted product, along with the product identifier. The product identifier is a file that contains information about the product, such as its description and its identifier. If the identifier of the encrypted product file corresponds to the identifier in the product identifier file, the transaction proceeds. If the

identifiers do not match, advice is send to the TTP and the customer waits for the correct encrypted product. This process ensures that the customer receives the product that was requested from the catalog. Next, the customer prepares a purchase order containing the customer's identity, the merchant identifier, the product identifier, and the product price. A cryptographic checksum is also prepared. The purchase order (PO), along with the cryptographic checksum, is then sent to the merchant. The combination of the PO and cryptographic checksum allows the merchant to ascertain whether the PO received is complete or whether it was altered while in transit. Upon receipt of the PO, the merchant examines its contents. If the merchant is satisfied with the PO, the merchant endorses the PO and digitally signs the cryptographic checksum of the endorsed PO. This is forwarded to the TTP. The TTP is involved in the process to prevent the merchant from later claiming non-acceptance of the terms and conditions of the transaction. The merchant also sends a single use decrypting key for the product to the TTP. Next, the merchant sends a copy of the encrypted product to the customer, together with a signed cryptographic checksum. The signed cryptographic checksum establishes origin of the product and also provides a check to signify whether the product has been corrupted during transit. Upon receipt of this second copy of the encrypted product, the customer validates that the first and second copies of the product are identical. Through this process customers can be assured that they received the product ordered. The customer then requests the decrypting key from the TTP. To do this the customer forwards to the TTP the purchase order and a signed payment token, together with its cryptographic checksum. The payment token contains the customer's identity, the identity of the customer's financial institution, the customer's bank account number with the financial institution, and the amount to be debited from the customer's account. To verify the transaction, the TTP first compares the digest included in the PO from the customer with the digest of the same from the merchant. If the two do not match, the TTP aborts the transaction. Otherwise the TTP proceeds by validating the payment token with the customer's financial institution by presenting the token and the sale price. The financial institution validates the token. If the token is not validated, the TTP aborts the transaction and advises the merchant accordingly. If the token is validated, the TTP sends the decrypting key to the customer and the payment token to the merchant, both digitally signed with the TTP's private key. Secure channels guarantee the confidentiality of all messages throughout this protocol. The protocol ensures money atomicity if the payment token generated by the customer contains the amount to be debited from the customer's account and credited to the merchant's account. Consequently, no money is created or destroyed in the system by this protocol. Goods atomicity is guaranteed if the TTP hands over the payment token only when the customer acknowledges the receipt of the product. The process also ensures that the product is actually available to the customer for use when the customer gives the go-ahead for payment by acknowledging the receipt of the good .Delivery verification is guaranteed if the TTP receives a cryptographic checksum of the product from the merchant. Also, the customer independently generates a checksum of the product received and sends it to the TTP. Using these two copies of the checksums, available at the TTP, both the merchant and the consumer demonstrate proof of the contents of the delivered goods.

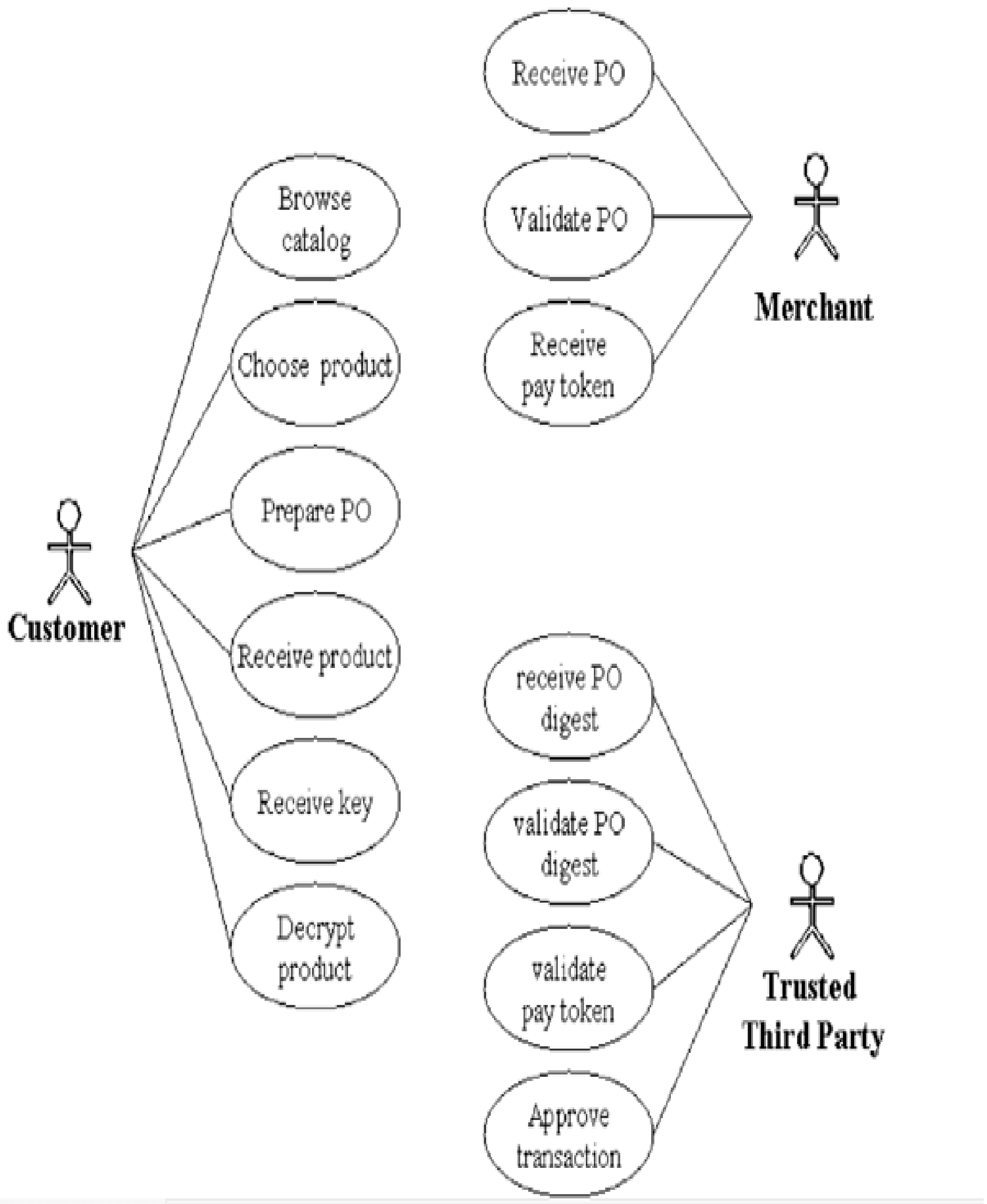

Fig. 1. High-level use-case diagram for trading digital products over the internet.

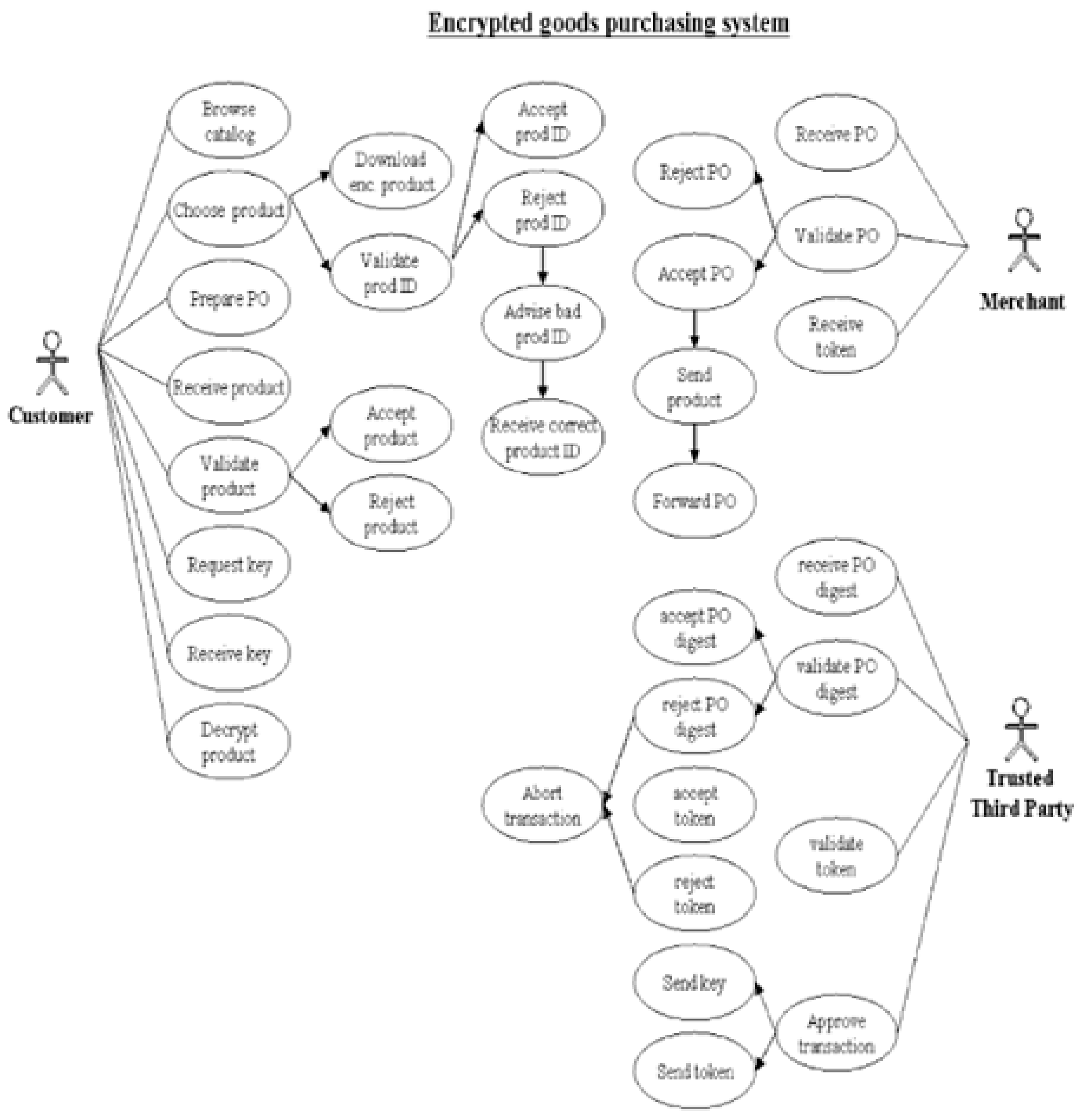

Fig. 2. Use-case diagram for comprehensive e-Business protocol.

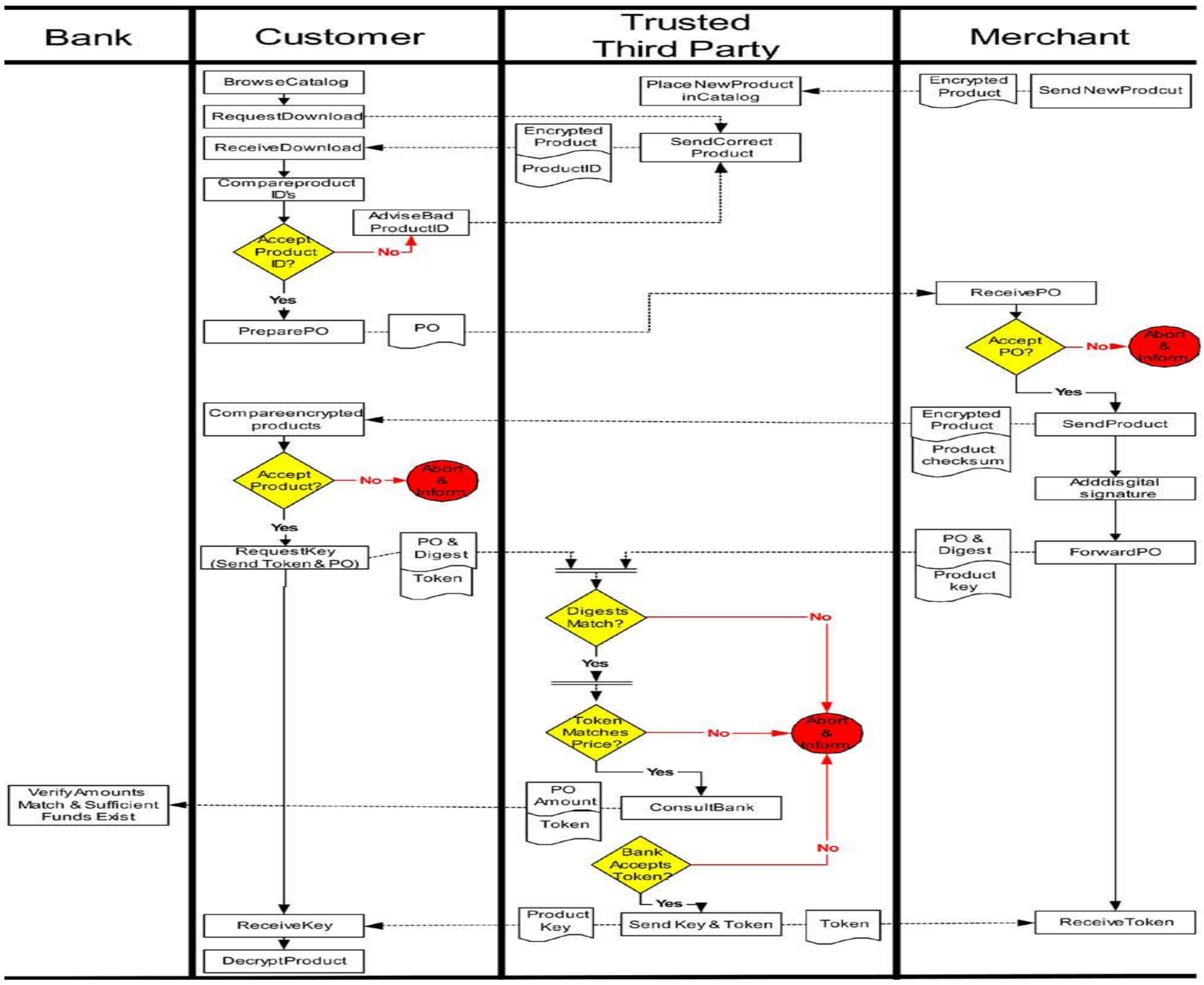

Fig. 3. Sequence diagram for comprehensive e-Business Protocol.

III. PROTOCOL IMPLEMENTATION

[2]This section discusses an implementation of the above protocol in FDR and an evaluation of its robustness.[4] In FDR model checking, which stands for ``Failures Divergence Refinement," the system model and the property specification are both state machines represented in the same language. The model checker then implements a refinement relation to see if the state space given by the model is a subset of the state space given by the property specification. Building FDR models of simplified versions of the NetBill and Digicash systems, which were then run through a model checker; we can referred the audience to the paper for the results, noting that while model checking has been useful for hardware verification, and recently also for software verification, this is the first time it has been applied to electronic commerce protocols. The FDR model implements key elements of the protocol with respect to money atomicity, goods atomicity, and valid receipt under several options. In order to avoid an overly technical presentation, the next section overviews a subset of representative processes that deal with money atomicity, goods atomicity, and validated receipt. The language of FDR is termed CSP (for communicating sequential processes). The writing of CSP code is greatly simplified by use of a compiler called Casper. Casper allows the user to describe the system in an abstract way, and the compiler converts that description to CSP code. We have included brief explanations of several expressions to assist the reader in understanding. These expressions represent processes that were outlined earlier, which should also aid in following the examples.

A. MODELING THE CUSTOMER PROCESS

The protocol starts when the customer browses the catalog hosted on the third party and downloads the encrypted product from there. The downloading of the encrypted product is modeled as the sending of the encrypted product by the third party and the receipt of the product by the customer. Thus, we can say that, initially the customer waits for an encrypted product from the third party.

CUSTOMER = cint? x -> DOWNLOADED_EGOODS(x).

Once the customer has downloaded the product, it sends a purchase order to the merchant. This is modeled as:
DOWNLOADED_EGOODS(x) = coutm ! po ->
PO_SENT(x)
The customer then waits for the encrypted product from the merchant. On receiving a message from the merchant, the customer checks to see if the message is indeed some encrypted product sent by the merchant. If so, the customer proceeds to the next step, otherwise it continues to wait for the encrypted product. The specification for this event is as:

PO_SENT(x) = cinm ?y ->
if (y==encryptedGoods1 or y==encryptedGoods2) then
RECEIVED_EGOODS(x,y)else PO_SENT(x)

The next step involves comparing the encrypted product received from the merchant with those downloaded from the third party. If the two do not match, the customer terminates the protocol.

RECEIVED_EGOODS(x,y)= if(x==y)then
RECEIVED_CORRECT_GOODS
else ABORT

When the customer is satisfied with the encrypted product, he sends the payment token to the third party.

RECEIVED_CORRECT_GOODS = coutt !paymentToken ->
TOKEN_SENT

After sending the payment, the customer waits for a message from the trusted third party. The third party either sends the customer the key or an abort message, depending on the outcome of the protocol. Once the customer has received the message from the third party, the protocol stops. Otherwise the customer continues to wait for the message.

TOKEN_SENT = cint ?y -> if (y==key) then SUCCESS else
if (y== transactionAborted) then ABORT else
TOKEN_SENT

B.MODELING THE MERCHANT PROCESS

On the merchant side, the protocol begins with the merchant waiting to receive a purchase order from a customer.

MERCHANT = minc ?x -> if (x==po) then PO_REC else
MERCHANT
The merchant in response must send an encrypted product to the customer. The merchant can act in two ways:

either he sends the correct encrypted
product (denoted by encryptedGoods1) or an incorrect
encrypted product (denoted by encryptedGoods2). This non-
deterministic choice is modeled as follows:

PO_REC = (moutc !encryptedGoods1 -
>ENCRYPTED_GOODS_SENT)|Ã‹|(moutc !
encryptedGoods2 -> ENCRYPTED_GOODS_SENT)
Once the merchant has sent the encrypted product, he must
send the decryption key to the trusted third party.
ENCRYPTED_GOODS_SENT = moutt !key -> KEY_SENT

After sending the key, the merchant waits to receive the payment token from the third party. The third party either sends the payment token or a transaction abort message if the transaction was aborted. The merchant process terminates once it receives a message, otherwise it continues to wait for the message.

KEY_SENT = mint ?x -> if (x==paymentToken)then
SUCCESS else if (x==transactionAborted)
then ABORT else KEY_SENT

C. MODELING THE TRUSTED THIRD PARTY PROCESS

The customer downloading the encrypted product, is modeled from the third partyâ„¢s end, as the trusted third party sending the encrypted product to the customer.
TP = toutc !encryptedGoods1 -> WAIT_TOKEN_KEY
The next step involves the third party waiting to receive the payment token from the customer and the key from the merchant. When the third party receives a message it checks if the message is a payment token or key or neither. Note that, it is not known whether the key or payment token will arrive first. If the payment token arrives first, the third party must wait for the key. On the other hand, if the key arrives first, the third party must wait for the payment token. This aspect of the protocol is modeled as follows:
WAIT_TOKEN_KEY = (tinc ?a -> if (a==paymentToken)then WAIT_KEY(a) else WAIT_TOKEN_KEY) [] (tinm ?b -> if (b==key) then WAIT_TOKEN(b)else
WAIT_TOKEN_KEY)WAIT_KEY(a)
=
tinm
?b
->
if
(b==key)
then
CHECK_TOKEN(a,b) else WAIT_KEY(a)
WAIT_TOKEN(b) = tinc ?a -> if (a==paymentToken) then CHECK_TOKEN(a,b) else WAIT_TOKEN(b)

Once the third party has received both the key and the payment token, it proceeds to the next step of validating the payment token with the customerâ„¢s financial institution. The details of the validation process is outside the scope of the protocol and is not modeled. Instead, the model no deterministically chooses between the options: (i) token okay or (ii) token not okay. If the payment token is okay, the third party proceeds to send out the key to the customer and the token to the merchant. If the payment token is not okay an abort message is sent to the customer and the merchant, and the protocol terminates.
CHECK_TOKEN(a,b) = OK_TOKEN(a,b) |Ã‹| NOK_TOKEN OK_TOKEN(a,b) = SEND_TOKEN_KEY(a,b) NOK_TOKEN = SEND_ABORT_MESSAGE
The process of sending an abortmessage to customer and merchant is modeled by the following step.
SEND_ABORT_MESSAGE = toutc !transAborted -> toutm !transAborted -> STOP
d.Modeling the Money Atomicity Property
Money atomicity is satisfied when one of the following things happen:(i)the customer sends the payment token and the merchant receives it or (ii) the customer sends the payment token and then receives a transaction abort message. This is modeled as
SPEC1 = STOP |Ã‹| ((coutt.paymentToken ->mint.paymentToken -> STOP) [] (coutt.paymentToken cint.transAborted -> STOP))
e.Modeling the Goods Atomicity Property
The goods atomicity property requires one of the following things to happen: (i) the customer receives both the correct encrypted product and the keys and the merchant receives the token, or (ii) the customer receives just the encrypted product and neither the merchant gets the payment token nor the customer the keys
SPEC2
=
STOP
|Ã‹|
((cinm.encryptedGoods1
->
STOP)[]
(cinm.encryptedGoods2 -> STOP) [](cinm.encryptedGoods1 ->
cint.key
->
mint.paymentToken
->
STOP)
[]
(cinm.encryptedGoods1 mint.paymentToken -> cint.key -> STOP))

f.Modeling the Validated Receipt Property
The validated receipt property ensures one of the following things happen:(i) the customer receives some encrypted product and does not make payment (either because he has received incorrect product or decides not to purchase the product), or (ii) the customer makes the payment after receiving the correct encrypted product. This is modeled as:
SPEC3
=
STOP
|Ã‹|
((cinm.encryptedGoods2
->
STOP)
[]
(cinm.encryptedGoods1->STOP)[](cinm.encryptedGoods1 coutt.paymentToken -> STOP))

Detecting Violation of Properties due to Failures
An informal analysis reveals that the properties may be violated if the customer, merchant, third party and Communication links fail arbitrarily. The following paragraphs describe how we use the model checker to detect failures that result in the destruction of the properties.
Introducing Unreliable Communication Channels The properties are violated when the following channels are made unreliable: (i) the channels connecting the third party and the customer and (ii)those connecting the third party to the merchant.

g.Introducing Failures in the Customer Process

In the following paragraphs we show how the properties get consider the first step for the customer process:

CUSTOMER = cint ?x -> DOWNLOADED_EGOODS(x)

Suppose we allow the customer to abort in this step. The question, then, is does any property get violated? To find out, we need to model the possibility of the customer aborting in the first step:

CUSTOMER = ABORT |Ã‹| (cint ?x -> DOWNLOADED_EGOODS(x))

The above specification says that the customer may abort or wait for the downloading of the encrypted product in a non-deterministic manner. After making the above alteration to the customer process, we use FDR to check for the satisfaction of the properties. As expected, the customer aborting in the first step, has no effect on the properties. We make similar modifications to each step in the customer process and check for the violation of the properties. Our results indicate that such modification to any step, except in the last step of the customer process (that is after the customer has sent the payment token), reserves all the properties. Allowing the customer to abort in the last step violates both money atomicity and goods atomicity.

toutc.encryptedGoods1,cint.encryptedGoods1,coutm.po,inc.po,moutc.encryptedGoods1,moutt.key,tinm.key,cinm.encryptedGoods1,coutt.paymentToken,tinc.paymentToken,toutc.transAborted,toutm.transAborted,mint.transAborted

The above sequence tells us that the following actions are executed. The customer downloads the encrypted product from the third party, then sends a purchase order to the merchant. On receiving the purchase order, the merchant sends the encrypted product to the customer and the key to the third party. The third party receives the key. The customer receives the encrypted product and validates it. The customer then sends the payment token to the third party. At this point, it appears that the customer aborts since we do not see any more messages sent or received by the customer. The third party receives the key from the merchant, the payment token from the customer, and then validates the token. The token turns out to be invalid and an abort message is sent by the third party to the customer and the merchant. Since the customer has aborted in the meantime, he does not get the transaction abort message from the third party. The merchant, however, receives the abort message. In the above scenario, the customer sends out the payment token, but neither the merchant received the payment token nor the customer the transaction abort message. Thus money atomicity is violated. Similarly, a counter example is generated illustrating how goods atomicity was violated.. Thus, our conclusion is that, the customer cannot abort after sending out the payment token and before receiving the key; if the customer does indeed abort we will no longer have money atomicity or goods atomicity. Failures in Merchant, Third Party Processes Allowing the merchant process to abort in the last step, that is, after sending the key but before receiving the payment token, violates both money atomicity and goods atomicity. Finally, we consider the third party process. The third party process can abort unilaterally only at its first step. Ensuring Failure Resilence of the Protocol From the above discussion we can summarize: (i) the customer cannot abort after he has sent the payment token to the third party. (ii) The merchant cannot abort after he has sent the product decryption key to the third party. (iii)The third party cannot abort unilaterally after its first step to ensure that the e-commerce protocol is resilient to site or link failures we propose the following extension to the basic protocol. We assume that each party involved in the transaction, keeps a copy of the information that it sends to another party for example purchase order, payment token and so on in its stable storage till such time as the information is no longer needed. Writes to the stable storage are atomic and durable until intentionally purged.

1. The customer, the merchant and the third party uses a system-wide unique identifier, Ti, to denote the current e-commerce transaction. The identifier is a tuple of the form <PID;C;M>, where PID is the identifier for the product the customer, C purchases from the merchant, M. The customer stores a log record of the form <Ti;INITIATE> to its stable storage and then sends the purchase order to the merchant.

2. When the merchant receives the purchase order, it writes a log record < Ti;INITIATE> to its stable storage; then the merchant checks to see if the purchase order is to its satisfaction. If it is not, the merchant writes an abort record in its log â€œ <Ti;ABORT> and aborts the transaction. It informs the customer of this decision. Otherwise it sends the encrypted product to the customer and the product decryption key and the approved purchase order to the third party. ininally, it writes a log record to its stable storage of the form < Ti;KEY-SENT>. At this stage the merchant enters a point of no return; it cannot abort unilaterally.

3. After receiving a message from the merchant the customer checks to see if it is an abort message or the encrypted product. If it is an abort, the customer aborts the transaction and writes a log record of the form <Ti;ABORT>. Otherwise the customer validates the encrypted product. If validated, the customer sends the payment token and purchase order to the third party and then writes a log record to its stable storage. The log record is of the form < Ti;PAYMENT- SENT>. This is the point of no return for the customer. If the encrypted product is not validated the customer can either request the product from the merchant, or abort the transaction.

4. One of the messages - either the message containing the payment token and purchase order from the customer or the message containing the product decryption key and approved purchase order from the merchant - will arrive at the third party before the other message. On receiving the message, the third party associates the unique identifier Ti to this current transaction and writes a log record to its stable storage of the form <Ti;INITIATE>. The third party starts a timer at this point. If the third party does not receive the other message before the timer expires, it writes a log record <Ti;ABORT> and sends abort messages to both the customer and the merchant.

5. After receiving the payment token from the customer, the third party validates the token with the customerâ„¢s financial institution. If the validation fails the third party writes a log record <Ti;ABOR> and informs both the customer and the merchant. Otherwise, after the third party has received both the product decryption key from the merchant and the payment token from the customer â€œ the third party sends the payment token to the merchant and writes a log record <Ti;PAYMENT â€œFORWARDED>, and sends the decryption key to the customer and writes a log record
<Ti;KEY-FORWARDED>.

6. The customer writes the log record <Ti;FINISH> after receiving the decryption key from the third party.

7. The merchant also writes a log record < Ti;FINISH>, after it has received the payment token.

Protocol Failure Analysis

1. Merchant fails after sending product decryption key but before writing <Ti;KEY-SENT>. After recovery from failure the merchant finds from its log that Ti has been initiated but the product decryption key has not been sent out (no information about the key having been sent is recorded). Consequently, it queries the third party to find out the status. If the status is abort, the merchant aborts. If the third party has not received the key, the merchant resends the key and write the appropriate record. If the third party cannot provide a status, the merchant resends the encrypted product to the customer, and the key and approved purchase order to the third party and writes the appropriate log records. It then waits for the payment token from the third party. Finally, as a result of the status query the merchant may receive the payment token. It then finishes by writing the appropriate log record.

2. Merchant fails after writing <Ti,KEY â€œSENT> or Merchant fails before writing <Ti,FINISH>. After recovery from failure, the merchant finds that it has not received the payment token. It asks the third party for the payment token. The third party responds either by sending the payment token or an abort message. If it is an abort message, the merchant write <Ti,ABORT> in its stable storage and aborts. If payment token is received the merchant writes < Ti,FINISH> to log.

3. Customer fails after sending payment token but before writing< Ti,PAYMENT â€œSENT>. After recovery, the customer notes from log that Ti has been initiated but no other information (such as, information about the product received or payment token sent) is recorded in the log. The customer, in this case, gets in touch with the merchant and asks for the product. The merchant either sends the encrypted product or an abort message. If the customer receives the encrypted product, the customer validates it, sends the payment token and writes the appropriate log record.

4. Customer fails after writing <Ti,PAYMENT â€œSENT> or Customer fails before writing<Ti,FINISH>. After recovery the customer notes that the decryption key has not been eceived. So it requests the third party for the product decryption key. The third part responds with either an abort message or the decryption key. If it is an abort message, the customer writes <Ti,ABORT> to its log and aborts. If it is the decryption key, the customer writes <Ti,FINISH> to the log and finishes.

5. Third party fails before writing <Ti,INIT IATE>. At this stage the third party is not aware of the transaction Ti. Consequently the third party does nothing. At some point of time either the customer or the merchant will get in touch asking for the product decryption key or a status query. At this stage the third party will write the log record <Ti,INITIATE> and ask the customer for the payment token and purchase order, and the merchant for the product decryption key and the approved purchase order. It then starts the timer.

6. Third party fails after writing hTi;INITIATEi. or Third party fails before writing either <Ti,PAYMENT FORWARDED> or <Ti,KEY FORWARDED>. After recovery the third party notes that Ti has been initiated. It asks the customer for the payment token and purchase order, and the merchant for the product decryption key and the approved purchase order. Once the third party has received a response, it starts the timer and waits for the other message.

7. Third party fails after writing one of the records<Ti,PAYMENT-FORWARDED> or <Ti,KEY FORWARDED> but before writing the other.After recovery the third party sends the message that was not sent out and writes the appropriate record.

IV. CONCLUSION

Model checking can be defined as, an approach based on exhaustive search of finite state spaces, could be applied to this system to verify its properties. A model of this system and a property specification could be given as input to a model checker, which would return a yes, meaning that the properties were verified, or provide a counter example .Model checking is an very powerful method for protocol verification. It is use to ensure an e-commerce protocol does satisfy the properties of money atomicity, goods atomicity, and validated receipt properties in the presence of site and communication failures. After verification, it proposes a mechanism that preserves the properties even in the event of sites or communications failures.

REFERENCES

[1] I. Ray, I. Ray, Failure analysis of an e-commerce protocol using model checking, Proceedings of the Second International Workshop on Advanced Issues of e-Commerce and Web-based Information Systems, Milpitas, CA, 2000 June.

[2] Anderson, B.B., Hansen,J.V., and Summers,S. Model checking for design and assurance of e-business process.Communications of ACM 49, 6 (June 2006),97-101

[3] http://seminarprojects.org/t-seminar-report-on-model-checking-for-securing-e-commerce-transactions

[4] Heintze, N., Tygar, J., Wing, J., and Wong,H. Model checking electronic commerce protocols I. Ray and I. Ray. Failure Analysis of an E-commerce Protocol using Model Checking . Technical report, University of Michigan-Dearborn, Jan. 2000.