An Efficient e-Commerce Fair Exchange Protocol That Encourages Customer and Merchant to Be Honest

Abdullah Alaraj and Malcolm Munro

Department of computer science
Durham University, the UK
{a.m.alaraj,malcolm.munro}@durham.ac.uk

Abstract. A new e-Commerce fair exchange protocol is presented in this paper. The protocol is for exchanging payment with digital product (such as computer software) between customer (C) and merchant (M). It makes use of Trusted Third Party (TTP) but its use is kept to minimum when disputes arise. In this respect it is an optimistic fair exchange protocol. A new idea, in which if the parties are willing to exchange then they are encouraged to be honest, is originated in this protocol. The protocol has the following features: (1) It comprises four messages to be exchanged between C and M in the exchange phase; (2) It guarantees strong fairness for both C and M so that by the end of executing the protocol both C and M will have each other's items or no one has got anything; (3) It allows both parties (C and M) to check the correctness of the item of the other party before they send their item; (4) It resolves disputes automatically online by the help of the Trusted Third Party (TTP); and (5) The proposed protocol is efficient in that it has a low number of modular exponentiations (which is the most expensive operations) when compared to other protocols in the literature.

1 Introduction

When exchanging digital products and payments over the internet, Merchants (M) and Customers (C) want to be sure that the exchange will be fair thus ensuring that C will receive the right product and M will get the correct payment. Fair exchange also means that if either or both parties are dishonest then both parties will receive each other's items or neither will receive anything.

There are three main processes involved in fair exchange of digital products and payments:

1. C sends the payment to M;
2. M send the digital product to C; and
3. Dispute resolution if something goes wrong.

Processes 1 and 2 can be carried out in either order, so C sends the payment to M and then receives the digital product or M sends the digital product to C and then receives the payment.

This paper will firstly discuss the literature on related work, and then present the details of the ECMH (Encouraging Customer and Merchant Honesty) protocol, and then evaluates the protocol by analysing different scenarios and comparing it against existing protocols.

M.D. Harrison and M.-A. Sujan (Eds.): SAFECOMP 2008, LNCS 5219, pp. 193–206, 2008.

2 Review of Literature

There is a number of fair exchange protocols described in the literature [5, 7, 10, 9, 3, 4, 1, 2, and 12].

Fair exchange protocols can be divided into two types: those that do not involve a trusted Third party (TTP); and those that do. This paper is concerned with the later type where the TTP takes all or some of the following roles: (1) ensures fairness in the exchange, (2) acts as certificate authority that is trusted by all parties, and (3) resolves disputes and/or validates items.

Protocols that involve a TTP can be of two types. The first type (such as [6]) uses a TTP for delivering the exchanged items. This involves each party sending their item to a TTP and the TTP delivering them to the parties. Involving a TTP will guarantee the fair exchange of items, but it has some drawbacks. The TTP could be the source of a bottleneck [10], it must always be available [8], and if the TTP crashes, the protocol will not deliver the items properly.

The second type is the protocols (such as [1, 2, 12, 5, 7, 10, 9, and 3]) where there is minimal use of the TTP, usually when something goes wrong. In these protocols the two parties directly exchange their items and the TTP only gets involved to resolve disputes. This type of protocol is called "Optimistic fair exchange protocols" [3].

The Ray et al [9] optimistic fair exchange protocol allows each party to verify whether the item they are going to receive from the other party is indeed the item they want, and this is done before receiving the item. A merchant (M) uploads the digital product to a TTP who encrypts it; the customer (C) downloads the encrypted digital product from the TTP to compare it with the digital product that will be received from M. The actual interaction between C and M in this protocol consists of four messages: C sends to M the first message which contains the purchase order and payment token that are encrypted; M sends a message to C which includes the encrypted digital product; C sends a message to M which includes the decryption key for the payment token; and finally M sends the decryption key of the product. If C has a dispute, then C contacts the TTP to resolve it. C needs to download the product twice (from TTP and then from M), so this will be a communication overhead.

Asokan et al [3] propose a generic optimistic fair exchange protocol that is suitable for exchanging signatures, confidential data or payments. If the items to be exchanged are payment and a digital product, the protocol can be explained as follows: a merchant (M) and a customer (C) promise to exchange their items using 2 messages; C sends the payment to M; M sends the digital product to C. If anything goes wrong then the TTP will cancel the payment. If the TTP is not able to do so then the TTP can provide an affidavit proof to be used in court to resolve the disputes. This protocol seems to be not the best solution for exchanging digital products fairly as TTP will not be able to resolve the disputes online.

The Zhang et al [12] fair exchange protocol for exchanging two valuable documents (the two documents can be a payment and digital product) comprises of four messages. The two parties involved in the protocol exchange their encrypted documents in the first two messages and then exchange the keys to decrypt them. The protocol is based on the idea of having each party verify the correctness of the key used to encrypt the document without seeing the key itself. The TTP can be contacted to recover the key if one party misbehaved.

The Alaraj and Munro [1, 15] protocol for exchanging digital products and payments consists of three messages to be exchanged between the customer (C) and the merchant (M). The messages are: M sends the encrypted digital product and its certificate to C; C verifies it and if satisfied sends to M the payment that is encrypted using a key that M already has; finally, the decryption key is sent to C by M when M is satisfied with the payment. If there is any dispute, the TTP will be contacted. This protocol enforces the customer to be honest because they cannot gain anything by being dishonest. Alaraj and Munro [2] extended the idea in this protocol to enforce the merchant to be honest. In this protocol C starts the exchange by sending an encrypted payment and its certificate to M, who verifies it and if satisfied, sends to C the digital product that is encrypted using a key that C already has. Finally, the decryption key is sent to M by C when C is satisfied. If there is any dispute, the TTP will be contacted to resolve the dispute.

In this paper, we applied the techniques used in [1, 15, 2] to encourage both C and M to be honest. This paper presents an optimistic fair exchange protocol for exchanging payment and digital product between customer and merchant. The proposed protocol allows both parties (customer and merchant) to check the correctness of the item of the other party before they send their items to them. Therefore, both parties are encouraged to be honest. The proposed protocol overcomes the drawbacks of the protocols in the literature.

3 Encouraging Customer and Merchant Honesty (ECMH) Protocol

3.1 Notations

This section defines the notation used in this paper, some of which are similar to the ones appear in [7].

- *C*: Customer
- *M*: Merchant
- *TTP*: Trusted Third Party which is a party neither M nor C that is trusted by all parties. TTP will not collude with any other party
- *D*: Digital product
- *CA*: Certificate Authority
- *CB*: the customer's Bank
- *desc.*: description of digital product
- *h(X)*: a strong-collision-resistant one-way hash function, such as SHA-1 [14]
- *pkx* = (ex, nx): RSA Public Key [13] of the party x, where nx is a public RSA modulus and ex is a public exponent
- *skx* = (dx, nx): RSA Private Key [13] of the party x, where nx is a public RSA modulus and dx is a private exponent
- *kx*: a symmetric key generated by x
- *P-Cert*: Payment Certificate that is issued by CB. *P-Cert* contents are:
    - *amount:* the amount of payment
    - *hP:* hash value of payment

  - *heP:* hash value of encrypted payment with *kc*
  - *heKc:* hash value of encrypted *kc*
  - *Sig.CB:* CB's signature on *P-Cert*
- *D-Cert*: Digital-product Certificate that is issued by CA. *D-Cert* contents are:
  - Price: price of D
  - *d*: Description of D
  - *hD*: hash value of D
  - *heD*: hash value of encrypted D with *km*
  - *heKm*: hash value of encrypted *km*
  - *Sig.CA*: CA's signature on *D-Cert*
- *C.mt*: the certificate for the shared public key between M and TTP; *C.mt* is issued by TTP. A standard X.509 certificate is used to implement *C.mt* [11]
- *C.ct*: the certificate for the shared public key between C and TTP; *C.ct* is issued by TTP. A standard X.509 certificate is used to implement *C.ct* [11]
- *enc.pkx(Y)*: RSA encryption of Y using the public key pkx (ex, nx). That is,

  enc.pkx(Y) = Y^{ex} mod nx = Z
- enc.skx(Z): RSA decryption of Z using the private key skx (dx, nx). That is,

  enc.skx(Z) = Z^{dx} mod nx = Y
- *enc.kx(Y)* : encryption of Y using a symmetric key kx (kx can also be used for decrypting enc.kx(Y))
- *Sig.A (X)*: RSA signature of the party A on X i.e. encrypting the hash value of X using the private key skA (dA, nA) as follows:

  Sig. A (X) = $(h(x))^{dA}$ mod nA
- A → B: X: A sends message X to B
- X + Y: concatenation of X and Y
- ECMH protocol: Encouraging Customer and Merchant Honesty protocol which is the protocol presented in this paper

3.2 Protocol Description

This protocol is for exchanging a digital product D with a payment. It is assumed that the payment in the protocol is in the form of a payment order that is issued and signed by a customer's bank and specifies the amount of payment to be paid, the payee and the payer. Double spending of the same payment is assumed to be detected and therefore will not occur. It is assumed that the communication channels between all parties (TTP, M and C) are resilient i.e. all sent messages will be received by their intended recipients [9]. C and M will agree on the TTP to be used in both the pre-exchange phase (by C) and the dispute resolution (by M) before they start the protocol.

The trustworthiness of C is governed by two things which are the payment certificate (P-Cert) issued by CB and the public key certificate (C.ct) issued by TTP. Therefore, the payment that will be sent by C is certified by CB; and the public key to be used by C to encrypt the key used to encrypt this payment is certified by TTP. The trustworthiness of M is also governed by two things which are the digital product certificate (D-Cert) issued by CA and the public key certificate (C.mt) issued by TTP. Therefore, the digital product that will be sent by M is certified by CA; and the public

key to be used by M to encrypt the key used to encrypt this digital product is certified by TTP. Therefore, this protocol encourages both C and M to be honest by sending correct items as each party will be able to detect if the received item is incorrect.

The scenario of this protocol is like C and M exchanging their encrypted items (payment and digital product) and their certificates. These encrypted items and their certificates will test the trustworthy of each party. If the parties found that the other party is trustworthy then they will complete the exchange otherwise they abort it.

3.2.1 Pre-exchange Phase

In the pre-exchange phase (Fig 1), C needs to get the certificate C.ct of the shared public key from TTP to be used to encrypt the key used to encrypt the payment (PE-b-M1 of Fig1). C also needs to get the payment and its certificate P-Cert from CB (PE-b-M2 of Fig1). The P-Cert is unique for each transaction (completed exchange) because the payment can only be used once. Also in the pre-exchange phase M needs to get the certificate C.mt of the shared public key from TTP to be used to encrypt the key used to encrypt D (PE-a-M1 of Fig1). M also needs to get the digital product (D) and its certificate D-Cert from CA (PE-a-M2 of Fig1), (the CA can be thought of as the producer of the digital product).

In this protocol, there are two public keys to be shared. The first one is shared between TTP and C. The other one is shared between TTP and M. The way in which these keys are shared is as follows.

- Each party (C, M and TTP) has its own public and private keys. The TTP's public key is denoted as pkt = (et, nt) and its corresponding private key is denoted as skt = (dt, nt). While C's public key is denoted as pkc = (ec, nc) and its corresponding private key is denoted as skc = (dc, nc); and M's public key is denoted as pkm = (em, nm) and its corresponding private key is denoted as skm = (dm, nm).
- The shared public key between C and TTP is denoted as pkct = (ect, nct) and its corresponding private key is denoted as skct = (dct, nct). The nct is a product of two distinct large primes chosen by TTP.

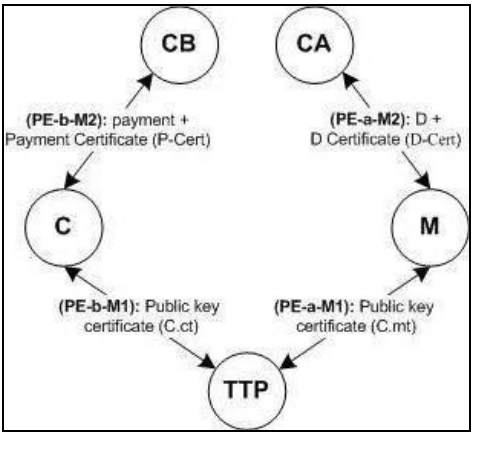

Fig. 1. Pre-exchange phase

- The shared public key between M and TTP is denoted as pkmt = (emt, nmt) and its corresponding private key is denoted as skmt = (dmt, nmt). The nmt is a product of two distinct large primes chosen by TTP

3.2.2 The Exchange Phase

It is assumed that the exchange phase will take place after C finds the wanted digital product (D) with M (either in M's website or through the search engines). It is also assumed that this phase will take place after C and M agree on the digital product and negotiated the price. Hence this phase is about the actual exchange of payment and digital product D.

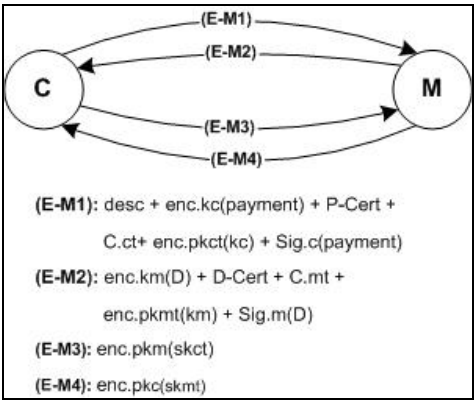

Fig. 2. The exchange phase

There are four messages to be exchanged between M and C in the exchange phase (Fig 2). These four messages are as follows.

[E-M1] **C → M: desc + enc.kc(payment) + P-Cert + C.ct+ enc.pkct(kc) + Sig.c(payment)**

C sends to M message E-M1 which contains the following:

- **desc:** specifies what C wants from M i.e. description of D that C wants (the description can be the digital product ID)
- **enc.kc(payment):** the payment that is encrypted with the key *kc*. *kc* is generated by C
- **P-Cert:** the payment certificate that is issued by CB
- **C.ct**: the shared public key certificate that is issued by TTP
- **enc.pkct(kc):** the key *kc* (that is used to encrypt the payment) is encrypted using the shared public key *pkct* that is certified in *C.ct*
- **Sig.c(payment):** C's signature on the payment. This signature can serve as non-repudiation of origin which allows M to be sure that the payment is sent by C. As explained in the notations section, C's signature on payment is the encryption of the hash value of payment using C's private key skc

***[E-M2]* M → C: enc.km(D) + D-Cert + C.mt + enc.pkmt(km) + Sig.m(D)**
On receiving message E-M1 from C, M checks the correctness of *enc.kc(payment)*, *enc.pkct(kc), P-Cert* and *C.ct*. The correctness of *P-Cert* can be checked by verifying CB's signature on *P-Cert*. Also the correctness of *C.ct* can be checked by verifying TTP's signature on *C.ct*.

To check that the encrypted payment is correct, M needs to check three things (1) the amount field in P-Cert against the price field in D-Cert that M has. This is to make sure that the payment meets the asked price; (2) the payment itself; and (3) the encrypted payment with *kc* i.e. *enc.kc(payment)*.

To check the correctness of payment, M needs to get the hash value of payment (*HP*) by decrypting *Sig.c(payment)* using C's public key *pkc* (the public keys of all parties are publicly available) and then compare it with hash value of payment (*hP*) that is included in *P-Cert*. That is, to check the following:

$$HP \; ?= \; hP$$

If they are the same then M can be sure that the actual payment is correct.

To check the correctness of the encrypted payment *enc.kc(payment)*, M computes the hash value of *enc.kc(payment)* (*HeP*) and then compare it with the hash value of encrypted payment with *kc* i.e. *heP* which is included in *P-Cert* (note that it is assumed that M will use the same function used by CB to compute the hash value). That is, to check the following:

$$HeP \; ?= \; heP$$

If they are the same then M can be sure that C encrypted the payment using *kc* and not another key.

M also needs to check the correctness of *kc* which is used to encrypt payment. To do so, M computes the hash value of *enc.pkct(kc)* (*HeKc*) and then compare it with *heKc* that is included in P-Cert, so M will check the flowing:

$$HeKc \; ?= \; heKc$$

If they are the same then M can be sure that the encrypted key is *kc* and not another key. The point here is to make sure that C is honest by sending the key used to encrypt the payment.

Therefore, if all comparisons are correct then, at this point, M will have the following fact. The encrypted payment is correct (i.e. it is the one described in *P-Cert*) and it is indeed encrypted with *kc*. In addition, the encrypted key in *enc.pkct(kc)* is indeed *kc* and not another key. The shared public key *pkct* used to encrypt *kc* is certified by TTP. Therefore, once M got the private key (*skct*) of the shared public key then M will be able to get the payment (by first decrypting *enc.pkct(kc)* to get *kc* and then decrypting *enc.kc(payment)* using *kc*).

Now, it is M's choice to complete the exchange or abort the protocol. If M wants to exchange D for the payment then M sends (in E-M2) the following:

- **enc.km(D):** the digital product D that is encrypted with the key *km* that is generated by M
- **D-Cert:** the digital product certificate that is issued by CA
- **C.mt:** the shared public key certificate that is issued by TTP

- **enc.pkmt(km):** the key *km*, that is used to encrypt D, encrypted using the shared public key *pkmt* that is certified in *C.mt*
- **Sig.m(D):** M's signature on D. This signature can serve as non-repudiation of origin which allows C to be sure that D is sent by M. As explained in the notations section, M's signature on D is the encryption of the hash value of D using M's private key skm

Note that if M decides to abort the transaction after receiving message E-M1 and before sending message E-M2 to C then neither M nor C lose anything.

[E-M3] **C → M: enc.pkm(skct)**
On receiving message E-M2 from M, C checks the correctness of *enc.km(D)*, *enc.pkmt(km), D-Cert* and *C.mt*. The correctness of *D-Cert* can be checked by verifying CA's signature on *D-Cert*. Also the correctness of *C.mt* can be checked by verifying TTP's signature on *C.mt*.

To check the correctness of D, C needs to check two things which are the digital product *D* itself and the encrypted D with *km* i.e. *enc.km(D)*. Firstly, to check the correctness of D, C needs to get the hash value of D (*HD*) by decrypting *Sig.m(D)* contained in message E-M2 using M's public key *pkm* (the public keys of all parties are publicly available) and then compare it with hash value of D (*hD*) contained in *D-Cert*. That is, to check the following:

$$HD \stackrel{?}{=} hD$$

If they are the same then C can be sure that the actual D is correct. Secondly, to check the correctness of the encrypted D *enc.km(D)*, C computes the hash value of *enc.pkmt(D)* (*HeD*) and then compare it with the hash value of encrypted D with *km* i.e. *heD* which is contained in *D-Cert* (note that it is assumed that C will use the same function used by CA to compute the hash value) i.e. to check the following:

$$HeD \stackrel{?}{=} heD$$

If they are the same then C can be sure that M encrypted D using *km* and not another key.

C also needs to check the correctness of *km* which is used to encrypt *D*. To do so, C computes the hash value of *enc.pkmt(km)* (*HeKm*) and then compares it with *heKm* that is included in D-Cert, so C will check the flowing:

$$HeKm \stackrel{?}{=} heKm$$

If they are compared then C can be sure that the encrypted key is *km* and not another key. The point here is to make sure that M is honest by sending the key used to encrypt D.

Therefore, if all comparisons are correct then, at this point, C will have the following fact. The encrypted D is correct (i.e. it is the one described in *D-Cert*) and it is indeed encrypted with *km*. In addition, the encrypted key in *enc.pkmt(km)* is indeed *km* and not another key. The shared public key *pkmt* used to encrypt *km* is certified by TTP. Therefore, once C got the private key (*skmt*) of the shared public key then C will be able to get D (by first decrypting *enc.pkmt(km)* to get *km* and then decrypting *enc.km(D)* using *km*).

Now, it is C's choice to complete the exchange or abort the protocol. If C wants to exchange the payment for D then C sends to M the decryption key *skct* encrypted using M's public key *pkm* to allow M be able to decrypt the encrypted payment.

Note that C must be sure that the encrypted D matches their requirements as explained earlier, otherwise C will be at risk if they send message E-M3 to M because when C sends to M the decryption key then this means that they are satisfied with E-M2 and hence M will be able to decrypt the payment.

Note that if C decides to abort the transaction after receiving message E-M2 and before sending message E-M3 to M then neither C nor M lose anything. But once C sends message E-M3 to M then the transaction must be completed and the protocol will guarantee that the exchange of payment and D will be fair if the sent items are as they described i.e. the payment matches the price that appears in message E-M2 and also the digital product matches *desc* that appears in message E-M1.

***[E-M4]* M → C: enc.pkc(skmt)**

On receiving message E-M3, M decrypts *enc.pkm(skct)* using M's private key *skm* to get the private key *skct.* Once M got *skct* then they decrypt *enc.pkct(kc)* to get *kc* that can be used to decrypt the encrypted payment received in E-M1.

If C encrypted the payment using different key (i.e. M was not able to decrypt the encrypted payment using *kc*) then M ignores the transaction and aborts the protocol. If however M managed to get the payment correctly then M sends to C in E-M4 the decryption key *skmt* that is encrypted using C's public key.

On receiving message E-M4, C decrypts *enc.pkc(skmt)* using C's private key *skc* to get the private key *skmt*. Once C got *skmt* then they decrypt *enc.pkmt(km)* to get *km* that can be used to decrypt the encrypted D received in E-M2.

If M encrypted D using different key (i.e. C was not able to decrypt the encrypted D using *km*) then C contacts TTP for resolution (as will be explained in the next section). If however C managed to get D correctly then the protocol finishes and the fair exchange of payment and digital product is ensured.

3.2.3 After Exchange (Dispute Resolution)

All disputes requests, if any, will come from C because M will not need to raise disputes as they get the decryption key of the encrypted payment and decrypt it before they send the decryption key of the digital product to C. Therefore, if C has a dispute, the following messages are executed (see Fig 3):

***[DR-M1]* C → TTP: D-Cert + C.mt + C.ct + enc.pkt(skct) + Sig.m(D)**

In case C has a dispute, they need to send to TTP the following: *D-Cert*, *C.mt, C.ct , enc.pkt(skct)* and M's signature on D that has been received in message E-M2 of the exchange phase.

***[DR-M2]* TTP → M: enc.pkm(skct)**

On receiving message *DR-M1* above, TTP will check the correctness of *D-Cert*, *C.mt, C.ct* by checking their signatures. If they are correct then TTP will decrypt the signature of M on D. That is, TTP decrypts *Sig.m(D)* to get the hash value of D included in the signature and then compares it with the hash value of D (*hD*) which is included in *D-Cert*. If TTP managed to decrypt *Sig.m(D)* correctly and the two hashes

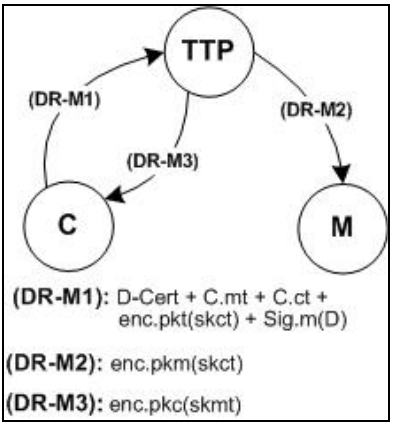

Fig. 3. Dispute resolution

are the same then TTP is sure that M was satisfied with the payment that C sent to them in message E-M1 of the exchange phase. This is because no other party can sign D as it needs M's private key which is only held by M. If M was not satisfied then they would not send *Sig.m(D)* to C in message E-M2. In other words, M will send message E-M2 (which includes *Sig.m(D)*) only if they are sure that the payment sent by C is correct. If TTP found the signature of M is correct then TTP sends to M the decryption key *skct* (encrypted with M's public key) to be used to get *kc* that decrypts the encrypted payment. The reason for sending the decryption key *skct* to M (as M is not the one who raises the dispute) is because C may have not sent the decryption key to M in message E-M3 or has sent incorrect decryption key.

Otherwise, if TTP found that the signature of M is incorrect then TTP sends an abort message to C and nothing will be sent to M.

[DR-M3] **TTP → C: enc.pkc(skmt)**
OR
TTP → C: aborts;

This is the same process for message *DR-M2* above, if TTP found that *Sig.m(D)* is correct then TTP sends to C the decryption key *skmt* (encrypted with C's public key) to be used to get *km* that decrypts the encrypted D. Otherwise if *Sig.m(D)* is incorrect then TTP sends an abort message to C.

It is clear that if either C has sent incorrect decryption key *skct* to M or C has not sent the decryption key at all in message E-M3 then C will not get an advantage over M because the TTP will check *DRM1* that C send to the TTP in order to check the signature of M. If the signature is correct then the TTP will send the decryption keys to both parties (C and M) to ensure fairness. Therefore, the fairness is ensured for both C and M. However, if the signature of M is incorrect then the TTP will reject C's request for the dispute.

As can be seen in the dispute resolution phase, the TTP does not need to have both C and M to be involved in order for the dispute to be resolved; rather only the disputant (C in this protocol) and the TTP will be involved. That is, the TTP does not need

to contact M to verify whether or not they have received the correct decryption key; rather TTP asks C to provide all evidences and finally will makes the resolution. M will only be contacted by the TTP if the dispute has a resolution. Therefore, this will reduce the number of messages needed to resolve dispute and as a result will reduce the load on the communication channels.

4 The Protocol Analysis

In ECMH protocol, the only party to raise a dispute is C. The following scenarios are presented and studied (Note that after C and M exchange their encrypted items in messages E-M1 and E-M2, they exchange the decryption keys):

- C received a correct decryption key, and M either received incorrect decryption key or has not received the decryption key at all. This case is not applicable in the ECMH protocol because C has to send a correct decryption key to M to be able to receive the correct decryption key from M
- C has either received incorrect decryption key or not received the decryption key at all, and M received the correct decryption key. In this case C will make a dispute to TTP as explained in section 3.2.3
- Both C and M have not received any decryption keys from each other. So, no dispute will be made as both of them have not revealed their items (the decryption keys). This represents the case where C received E-M2 and did not send E-M3 to M or the case where C sends E-M1 to M but M does not send E-M2 to C
- Both C and M have received incorrect items (decryption keys) from each other. That is, C received incorrect decryption key and M received incorrect decryption key. This case is not applicable in the ECMH protocol because C has to send a correct decryption key to be able to receive the correct decryption key from M. So, if M found that the decryption key is incorrect then M will not send to C neither correct the decryption key nor incorrect decryption key
- C received incorrect decryption key and M has not received the decryption key at all. This case is not applicable in the ECMH protocol because C has to send a correct decryption key to M to be able to receive the correct decryption key from M. So, if M has not received the decryption key then M will not send the decryption key at all
- C has not received the decryption key at all and M received incorrect decryption key. This case is normal to occur because if C sent incorrect decryption key then M will not send their decryption key to C. Therefore, if this case occurs then for C to raise a dispute to the TTP, C needs to send to the TTP a correct DR-M1 (see message DR-M1 in section 3.2.3). If C sends the correct DR-M1 to the TTP then the TTP will make a resolution to both C and M. However, if the TTP found that DR-M1 is incorrect then C's dispute will be rejected

It is clear how the design of the ECMH protocol reduces the possibilities for having disputes. Additionally, in the ECMH protocol only C will raise disputes as M will not send their item unless the item of C is correct. As a result, the possibilities for disputes are reduced by preventing them.

In addition to the previous cases, the following cases (scenarios) are studied:

- C disputes to the TTP that they have received incorrect digital product: this scenario is not possible because *D-Cert* guarantees that the digital product is correct; and if C found that the digital product is incorrect or not the same as they wanted then they should have not sent to M the decryption key in E-M3. So, it is C's fault to send to M the decryption key if they have a doubt about the digital product. But once C sends to M the decryption key then this means that they are satisfied with the digital product. Therefore, this scenario will not happen because C knows the rules of the protocol which allow C to check the digital product before they send the decryption key to M; and as a result C will not put themselves at risk
- It is clear that M will not raise a dispute because M will receive from C the decryption key *skct* and get the payment before they send the decryption key *skmt* to C. However, the following scenarios are studied:
  - M claims that they have received incorrect payment from C: this will not occur because if M received incorrect payment then they will not send the encrypted digital product in message E-M2 and hence no one will get advantage over the other party. However, once M sends message E-M2 to C then this means that they are satisfied with the encrypted payment
  - M claims that they have not received the decryption key *skct*: this is not applicable in this protocol because if the decryption key is not received then no party is hurt and the fairness is not compromised. The reason for not receiving the decryption key *skct* may be because C is not satisfied with the encrypted digital product
  - M claims that they have received incorrect decryption key *skct* from C: this is not applicable in this protocol because if the decryption key is incorrect then no party is hurt and the fairness is not compromised as if the *skct* is incorrect then M will not their decryption key (*skmt*) to C.

5 Comparisons

The ECMH protocol presented in this paper has been compared to some of the protocols described in the literature that have the same characteristics in that they are used for exchanging digital products and payments and are based on RSA [13]. Thus ECMH protocol is compared to Ray et al [9] (denoted as Ray protocol) and Zhang et al [12] (denoted as Zha protocol).

The comparisons are made using the following criteria. (1) number of messages in both the exchange and dispute resolution phases, (2) whether or not the TTP needs to hold a copy of an item to be exchanged, (3) whether or not all parties (M and C) will be involved to allow the TTP to resolve any disputes, and (4) number of modular exponentiations in both the exchange and dispute resolution phases. The modular exponentiations are considered to be the most expensive operation [7].

The Ray et al [9] paper did not give details of the dispute resolution phase so the number of messages and the number of modular exponentiations had to be estimated manually. In addition, the number of modular exponentiations for Zha's protocol has also been estimated manually.

As can be seen in Table 1, all protocols have the same number of messages between C and M in the exchange phase. Ray's protocol lets the TTP hold M's item before the exchange between C and M takes place. This requires more storage and security assurance to be added to the TTP's jobs. Additionally, this may compromise the confidentiality of the items to be exchanged.

The Ray protocol requires both parties (C and M) to be contacted by the TTP when one party raises a dispute; whereas in the ECMH protocol and the Zha protocol only the disputant and the TTP will be involved. Involving both parties in dispute resolution would require more messages to be sent and hence more load on the communication channels.

The ECMH protocol has the lowest number of modular exponentiations needed to generate and verify messages in the exchange phase. While ECMH protocol has more modular exponentiations in the dispute resolution phase. However, most of modular exponentiations in ECMH protocol are for adding more security assurances such as encrypting the content of messages to prevent any other party (not those involved in the protocol) from gaining any useful information. This means that, 4 out of 14 modular exponentiations in the exchange phase and 6 out of 9 modular exponentiations in the dispute resolution phase are for adding such security assurances. The implication of this is that if there is an assumption that the channels are secured then the number of modular exponentiations are 10 and 3 for the exchange phase and dispute resolution phase, respectively.

Table 1. Protocols comparisons

	Ray protocol [9]	Zha protocol [12]	ECMH protocol
# messages (exchange phase)	4	4	4
# messages (dispute resolution)	3 to 5	3	3
TTP hold item	Yes	No	No
Both parties are involved in dispute resolution	Yes	No	No
# modular exponentiations (exchange phase)	27	20	14
# modular exponentiations (dispute resolution phase)	5 to 6	6	9

6 Conclusion

A new fair exchange protocol for exchanging digital products and payment has been presented in this paper. It comprises four messages to be exchanged between C and M. The protocol uses certificates that are issued by trusted parties such as a TTP, a CA and a CB. These certificates are *P-Cert* which allows M to check the correctness of *payment*, *D-Cert* which allows C to check the correctness of *D, C.mt* which allows C to check the origin of the key used to encrypt the key used to encrypt D and *C.ct* which allows M to check the origin of the key used to encrypt the key used to encrypt the *payment*. The only way in which M might misbehave after receiving the decryption key from C is by sending incorrect decryption key or by not sending it at all. This

can be resolved automatically and online by the help of TTP. The protocol guarantees strong fairness for both C and M.

References

1. Alaraj, A., Munro, M.: An e-commerce Fair Exchange Protocol for exchanging Digital Products and Payments. In: Proceedings of IEEE ICDIM 2007, Lyon, pp. 248–253 (October 2007)
2. Alaraj, A., Munro, M.: An Efficient Fair Exchange Protocol that Enforces the Merchant to be Honest. In: Proceedings of IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing 2007, CollaborateCom 2007, New York, pp.196–202 (November 2007)
3. Asokan, N., Schunter, M., Waidner, M.: Optimistic Protocols for Fair Exchange. In: Proc. Fourth ACM Conf. Computer and Communication Security, Zurich, Switzerland, pp. 8–17 (April 1997)
4. Ben-Or, M., Goldreich, O., Micali, S., Rivest, R.: A Fair Protocol for Signing Contracts. IEEE Trans. Information Theory 36(1), 40–46 (1990)
5. Ezhilchelvan, P., Shrivastava, S.: A Family of Trusted Third Party Based Fair-Exchange Protocols. IEEE transactions on dependable and secure computing 2(4) (October-December 2005)
6. Ketchpel, S.: Transaction Protection for Information Buyers and Sellers. In: Proceedings of the Dartmouth Institute for Advanced Graduate Studies 1995: Electronic Publishing and the Information Superhighway, Boston, USA (1995)
7. Nenadic, A., Zhang, N., Cheetham, B., Goble, C.: RSA-based Certified Delivery of E-Goods Using Verifiable and Recoverable Signature Encryption. Journal of Universal Computer Science 11(1), 175–192 (2005)
8. Pagnia, H., Vogt, H., Gärtner, F.: Fair Exchange. The Computer Journal 46(1) (2003)
9. Ray, I., Ray, I., Narasimhamurthy, N.: An Anonymous and Failure Resilient Fair-Exchange E-Commerce Protocol. Decision Support Systems 39, 267–292 (2005)
10. Ray, I., Ray, I.: An Optimistic Fair Exchange E-Commerce Protocol with Automated Dispute Resolution. In: Bauknecht, K., Madria, S.K., Pernul, G. (eds.) EC-Web 2000. LNCS, vol. 1875, pp. 84–93. Springer, Heidelberg (2000)
11. Public-Key Infrastructure (X.509), The PKIX working group (accessed on 08-06-2007), `http://www.ietf.org/html.charters/pkix-charter.html`
12. Zhang, N., Shi, Q., Merabti, M., Askwith, R.: Practical and Efficient Fair Document Exchange over Networks. The Journal of Network and Computer Applications, the Elsevier Science Publisher 29(1), 46–61 (2006)
13. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM, 120–126 (1978)
14. Ferguson, N., Schneier, B.: Practical cryptography. Wiley, Indianpolis (2003)
15. Alaraj, A., Munro, M.: An e-Commerce Fair Exchange Protocol that Enforces the Customer to be Honest. International Journal of Product Lifecycle Management, IJPLM (to appear)