Received May 18, 2020, accepted May 30, 2020, date of publication June 8, 2020, date of current version June 18, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3000558

Removing the Trusted Third Party in a Confidential Multiparty Registered eDelivery Protocol Using Blockchain

MACIÀ MUT-PUIGSERVER, MIQUEL A. CABOT-NADAL, AND M. MAGDALENA PAYERAS-CAPELLÀ
Departamento de Ciències Matemàtiques i Informàtica, Universitat de les Illes Balears, E-07122 Palma, Spain

Corresponding author: Macià Mut-Puigserver (macia.mut@uib.cat)

This work was supported in part by the Ministerio de Economia, Industria y Competitividad (MINECO), in part by the Agencia Estatal de Investigacion (AEI), and in part by the European Regional Development Funds (ERDF) through the Project Fair Exchange, Loyalty and TIckets with blockCHAIN (FeltiCHAIN) under Grant RTI2018-097763-B-I00.

ABSTRACT Recently several proposals of blockchain-based solutions for traditional e-commerce applications have been presented, taking advantage of the fact that blockchain is a technology that offers an immutable registry of data. Among these proposals we can find solutions for certified notifications, digital signature of contracts, escrow protocols, fair payments and registered deliveries. In order to execute fair exchanges, most solutions involve trusted third parties, known as TTP, supervising the exchanges in a way or another. Until now, two solutions have been presented for Registered electronic Delivery (eDelivery) services. This service allows a user to prove that he has sent some data to a set of receivers. These protocols differ in the properties achieved and also in the use of trusted third parties. The first protocol is a blockchain-based solution without TTP for the eDelivery of non-confidential data. The second protocol allows also the eDelivery of confidential data. However, this second proposal requires the involvement of a TTP in a non-mandatory resolution phase. In this paper we present a new protocol that achieves the best properties of the previous solutions at the same time. The new protocol doesn't require the involvement of a TTP at any moment while it allows the eDelivery of confidential data, satisfying the security requirements for this service.

INDEX TERMS Blockchain, notification, smart contract, confidentiality, fairness, cryptocurrencies.

I. INTRODUCTION

Recently several proposals of blockchain-based solutions for traditional e-commerce applications have been presented. They take advantage of the fact that blockchain is a technology that offers an immutable data registry. Among these proposals we can find in the literature solutions for certified notifications, digital signature of contracts, escrow protocols, fair payments and registered deliveries. In [20] two blockchain-based approaches for Registered electronic Delivery (eDelivery) services were presented. This service provides to the users a sender's proof of having sent a message via an electronic mailing system against an electronic verification that the message was delivered or that a delivery attempt was made (i.e. a non-repudiation of origin evidence against a non-repudiation of reception evidence).

Registered delivery services are mainly offered by postal services in many countries and they have different denominations depending on each service provider. For instance, most postal services offer a Registered mail service that include the sending of *lettermail*, documents and valuables. In the case of the United State Postal Service (USPS), the service is called Certified Mail, but it also offers a Registered Mail service that additionally provides the Chain of custody properties. That is, the collection of information that provides evidence about the chronological actions in the delivery service or sequence of custody, control, transfer, analysis, and disposition of the delivery. Also, the Registered Mail service of the USPS can

The associate editor coordinating the review of this manuscript and approving it for publication was Shadi Aljawarneh.

specify the delivery status or attempted delivery status when the item reaches its destination.[1] This way, this kind of services provide evidence that a user that acts as a receiver (or set of receivers) has access to the data since a specific instant. As trust services, these proposals must offer a high level of security and protection of the privacy of the users but they also have to consider the regulations on the subject. Therefore, distributed ledger, data protection and immutability features of the blockchain technologies make the blockchain an ideal tool to offer trust and data trail for new eDelivery solutions.

It is usual that e-commerce users exchange data or items among them. Registered eDelivery services, together with other e-commerce services, like electronic purchases and electronic signature of contracts, require the fair exchange of those data or items. A fair exchange aims to provide an equal treatment to all the involved parties. At the end of a protocol execution, either each party has the item it desires to obtain from the other involved party, or, if it is not the case, the exchange has not been performed successfully for any user, that is, any user has not received the desired element. In the design of these protocols a method that allows to perform the exchanges and assure the security of the exchange is required.

In order to execute fair exchanges, most solutions include trusted third parties that manage the exchanges with more or less involvement. The TTPs are responsible for the resolution of all the conflicts that can arise among the parties as a result of a non concluded exchange or a fraud attempt. Current fair exchange protocols, as [11], [12], [21] involve TTPs in several degrees, with similar functions to those of a judge or notary. Nevertheless, the acceptance of TTPs can be an obstacle to generalize the use of this kind of protocols. First, it is difficult to have really reliable TTPs for any user and, in addition to that, we have to take into account that they must be useful in different scenarios (e.g. electronic documents generated by TTPs must be accepted to resolve disputes in courts of law of different countries). Then, TTPs could cause technical problems (e.g., bottlenecks), reduce the efficiency of the protocols (e.g., delays in the resolution of conflicts) and they also increase the execution costs (e.g. high service rates). Moreover, they are a very sensitive point because the security of the exchange could be compromised if the TTP has any vulnerability.

In the protocols presented in [20] for eDelivery services, the elements to exchange are the data to be delivered along with non-repudiation of reception and origin proofs. The two protocols presented in [20] differ in the properties achieved and also in the use of trusted third parties. The first protocol is a blockchain-based solution without TTP for the eDelivery of non-confidential data. The second protocol allows also the eDelivery of confidential data. However, this second proposal requires the involvement of a TTP in a non-mandatory resolution phase. Since the publication of [20] no new protocols for confidential eDelivery without TTP have been proposed.

[1](https://faq.usps.com/s/article/What-is-Registered-Mail#international)

In this paper we present a new protocol that achieves the best properties of the previous solutions at the same time, avoiding the need to choose one property and renounce to the other. It doesn't require the involvement of a TTP at any moment while it allows the eDelivery of confidential data.

The paper is organized in the following sections. After this introduction, we review the desired properties of Registered eDelivery systems and also we present the state of the art on this topic in Section § II. The contribution of this research is described in Section § III. Then, the system overview is depicted in § IV while § V contains the description of the protocol for blockchain-based multiparty confidential eDelivery. Section § VI describes the implementation and the Smart Contracts of the protocol, including the implementation of the zero knowledge proof. Then an analysis of the protocol is performed; Section § VII presents the analysis of the ideal properties for this application while Section § VIII includes an exhaustive analysis of the performance of the protocol in terms of both cost. At the last, the conclusions are enumerated in Section § X.

II. PROPERTIES AND STATE OF THE ART

Sets of ideal properties related to registered eDelivery services can be found in several documents, both from the technical and the legal point of view. From the legal point of view, we have evaluated the document [23], that lists the ideal features of a registered eDelivery service, organized into security, legal and functional features. These requirements are related with the data integrity, the delivery of the data by an identified user, the reception by an identified addressee and the timestamping of sending and reception. The aforementioned features have to be taken into consideration to make a list of properties that should fulfill an eDelivery system. Moreover, the technical requirements for fair exchange protocols were stated in [1] and in [24]. Consequently, we have classified and summarized such ideal properties, since registered eDelivery service is a special case of a fair exchange of values. We presented the ideal properties of a registered eDelivery system in [20]. The most important properties, that will be used in this paper are:

1) **Effectiveness.** If the parties behave correctly, they will receive the expected items.
2) **Fairness.** After completion of a protocol execution, either each party has received the expected item or neither party has received any useful information about the other's element.
3) **Timeliness.** At any time, during a protocol execution, each party can unilaterally choose to terminate the protocol without losing fairness. In addition to that, a protocol can achieve *Weak Timeliness* if any honest party can be sure that any protocol execution will be concluded at a certain finite point in time. That is, the state of the exchange will be final from the party's perspective at this completion point.

4) **Non-repudiation.** If an element has been sent from party A to party B, A cannot deny the origin of the element and B cannot deny receipt of the element.
5) **Confidentiality**. Only the sender and the receiver of the data know the contents of the registered message.
6) **Efficiency.** An efficient protocol uses the minimum number of steps that allows the effective exchange or the minimum cost.
7) **Transferability of evidence.** The proofs generated by the system can be transferred to external entities to prove the result of the exchange.

Since the protocol proposed in this paper does not require a TTP, the properties related with the behavior of the TTP (transparency, verifiability,...) are not included.

As it is stated in [20], eDelivery follows the pattern of fair exchange of values. This kind of exchange does not have a definitive and standardized solution in its electronic version. The notification of a message can be done using electronic mail and, until now, several proposals have been presented for this service. However, it is not required that the eDelivery uses electronic mail. It includes an exchange of elements between the sender and one or several receivers in the multiparty case; the sender has to send a message to the receiver or set of receivers, then the receivers are able to read it and, in exchange, the receivers have to send a proof of reception to the sender. To overcome reluctance between the parties and to assure fairness, almost all the existing proposals use a TTP. This trusted third party can play and important role, participating in each exchange, or a more relaxed role in which the TTP is only active in case a dispute arises between the parties (optimistic protocols) [1].

Previous studies on fairness using blockchain [6], [10] focus on exchanges including payment, fair purchase operations between a product (or a receipt) in exchange for cryptocurrencies (usually bitcoin). Reference [15] uses, for the first time, a smart contract for the resolution of a fair purchase operation. In [3] Delgado-Segura *et al.* propose a protocol based on Bitcoin scripting language for a fair exchange between payment and data.

In [9], Hasan *et al.* propose a non-repudiation protocol using the blockchain technology and the Ethereum smart contracts. Exposito *et al.* in [29] describe a possible blockchain notification system for mobile apps. The system is applied to event-based subscriptions supported by a blockchain infrastructure deployed as a cloud service, however the concision of the proposed system is not clear and no smart contracts are proposed to secure the messaging scheme. Also, Zupan *et al.* in [30] propose a notification system based on smart contracts deployed on a blockchain using Hyperledger Fabric. Although the system provides authentication via certificates issued by Fabric's CA, this scheme does not provide neither a fair exchange scheme of delivery notifications nor proof of reception of the issued information.

We have published [16], [19], [20] several works focusing on the incorporation of blockchain in certified notification protocols and a paper proposing a blockchain-based protocol for contract signing operations [17]. As far as we know, there are no other works that deal with blockchain-based registered eDelivery services using smart contracts. Reference [16] presents two proposals, one of them enables a non-confidential fair exchange of a notification message for a non-repudiation of reception token with no involvement of any TTP. The second one allows a confidential fair exchange of a notification for a non-repudiation of reception token. The other one has the optimistic involvement of a stateless TTP. Moreover, in [19] we introduced the use of reusable smart contracts for several notifications. A multiparty eDelivery [18] allows a sender to send, in an efficient way, a message to multiple receivers. [20] proposes a multiparty protocol for non-confidential notification with no involvement of any TTP and a multiparty protocol that has the optimistic involvement of a stateless TTP.

III. CONTRIBUTION

A. BRIEF REVIEW OF PREVIOUS PROPOSALS

In [20], two protocols were presented. Each one of them satisfied interesting requirements. While the first proposal allowed the complete execution of the delivery or notification operation without the implication of any trusted third party for non-confidential eDeliveries, the second proposal allowed confidential eDeliveries thanks to the possible use of a trusted third party. Thus, to send a notification, users first had to select if they would like to send a secret or a public notification. In both cases, sender and receivers run a three-step exchange to transmit the content of a eDelivery together with non-repudiation evidence.

We can summarize the protocols as follows:

- First step: the sender presents a new eDelivery and submits it in a hidden mode to all recipients. Some parameters are incorporated in this step to guarantee the fairness property of the exchange, related to the content of the eDelivery and the non-repudiation evidence.
- Second step: Every recipient is able to decide whether he wants to receive or not the eDelivery. The ones who had accepted the message have to issue a non-repudiation of reception evidence.
- Third step: the sender is able to finish the protocol providing a way to get the plain content of the eDelivery and the non-repudiation of origin evidence for all recipients who had accepted the notification.

The execution of three-step protocol in the confidential scheme was *off-chain*, since sender and recipients exchanged messages directly. Only, in case of problems, to assure the security properties, the sender was able to cancel the exchange using a smart contract deployed specifically for this issue. Also, if receivers could not successfully conclude the exchange, they were able contact with the TTP that will check the status of the eDelivery with the assistance of the smart contract and, depending on the result, it will issue alternative evidence to guarantee fairness. Therefore, an additional

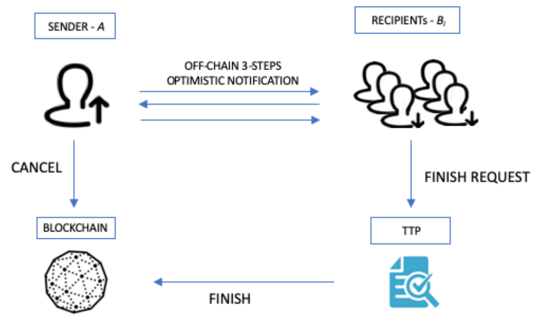

FIGURE 1. Interaction among the actors in the Confidential Protocol with TTP.

conflict resolution protocol was specified between recipients and TTP to solve the exception cases. The actions of the TTP were recorded on the blockchain by means of the smart contract (the interaction of the confidential protocol is depicted in Figure 1).

Whereas, in the non-confidential eDelivery protocol, all users execute the three-step procedure *on-chain*, they call the smart contract functions to perform and provide evidence of each action.

B. NEW PROPOSAL

This new proposal achieves the fulfillment of the best part of each one of the previous protocols in a single protocol. This way, the protocol allows the delivery of confidential data without the need of trusted third parties.

In order to achieve confidentiality in an exchange that is publicly managed by a smart contract and, as a method for the improvement of the previous proposal, we have determined that the protocol has the following requirements:

- The delivered data must be encrypted until the acceptation by the receiver, when the non-repudiation of reception proof is provided by the receiver.
- The smart contract cannot access the key required to decrypt the delivered data. The key cannot be included in clear in a transaction.
- The smart contract must assure that the receivers will be able to decrypt the data after acceptation of the delivery.
- Since the protocol is multiparty, the smart contract must assure that all the receivers decrypt the same data.

These requirements must be achieved together with the requirements to obtain fairness and the other desired properties.

Next, we present the protocol, the description of the cryptographic operations used in it and the implemented application with the smart contracts. The new protocol will be analyzed both in terms of performance and security.

IV. SYSTEM OVERVIEW

In this proposal, confidentiality is required. The solution provides fairness to the exchange of elements: message and non-repudiation proofs even when the smart contract does not know the content of the delivered data and the plain message is not registered on the blockchain. The sender, A, executes the first step of the protocol by means of the DApp in order to register the encrypted data on the blockchain. The encryption has to guarantee that only the receivers can get access to the content of the notification. Moreover, due to the nature of the multiparty notification service, this step has to be designed in a way that the smart contract execute a verification to assure that the message that each receiver can decrypt has the same content. To execute this step, all users have to generate a pair of keys, which will be called *Notification Keys*. The receivers, members of B, have to accept the notification by means of a transaction. The transaction is stored in the blockchain. Finally, A will execute a new transaction finishing the exchange in the third step of the protocol.

The cryptographic algorithms used in the design of the protocol are:

- ElGamal Encryption. The encryption and decryption processes are performed *off-chain*.
- Schnorr Zero-Knowledge proofs (ZKP). The verification of the ZKP is performed *on-chain* by the smart contract. The description of the solidity implementation of the ZKP is included in this paper.

The ZKP proof is introduced in the protocol to check whether all receivers are able to decrypt the same notification content or not and, at the same time, the scheme is preserving the confidentiality of the delivered data. In this way, the sender commits to send a key to all the receivers during the creation of a new eDelivery. This key will allow each receiver, who has accepted the message, to open its content. Thus, the key, instead of being published, is encrypted with every *secret shared notification key* that it is only known by each receiver and the sender. The ZKP introduced in the protocol allows the smart contract to check that the key to open the message is the same for all the receivers without knowing the bit string of the key. Therefore, the protocol preserves the confidentiality between sender and receiver and, at the same time, it can publicly verify that each receiver has access to the same content.

V. CONFIDENTIAL BLOCKCHAIN-BASED MULTIPARTY REGISTERED eDelivery PROTOCOL WITHOUT TTP

A. CRYPTOGRAPHIC BACKGROUND AND NOTATION

Proper parameters should be published prior to the use of the protocol. Also, users have to be able to check the correctness of these domain parameters. The appropriate operational conditions, before starting with the *Creation* phase of the protocol as it is described in next section, are the following:

- Two large primes have to be published: p and q with $q|p-1$ (q is a primer large factor of $(p-1)$)
- The operations will be made in $GF(p)$ and in Gq. Where Gq denotes the subgroup of the multiplicative group of $GF(p)$, of prime order q
- Let g be a generator for the subgroup Gq, such that $1 < g < p$
- Random numbers r and s are uniformly chosen between 0 and $(p-1)$

TABLE 1. Notation for the blockchain-based Registered eDelivery protocol.

Notation	
A	Sender.
B	Set of receivers. B_i is used for a single receiver.
B'	Set of receivers that have accepted the delivery
M	Message, content of the eDelivery.
$M[j]$	Fragment of the message.
X, Y	Concatenation of messages X and Y.
$U \blacktriangleright e.f$	Execution of function f of e by user U.
$term_1$	Timeout for B_i to accept the eDelivery.
$term_2$	Timeout for A to finish the exchange.
D	Deposit sent to the Smart Contract (ethers).
x_A	A's notification private key.
y_A	A's notification public key.
x_{Bi}	B_i's notification secret shared key.
y_{Bi}	B_i's notification public key.
g, p, q, g	System parameters.
r	Encryption secret nonce used by A.
s	Encryption nonce used by B.
$C_1 = g^r_{modp}$	First part of ElGamal encryption of the data to deliver.
$C_2[j] = M[j]$ XOR key	Second part of ElGamal encryption of the data to deliver.
$key = random.seed(hash(r))$	Encryption key
$h()$	Hash Function
$challenge$	Challenge sent by B to A
w	Response to the challenge.
$Z_{i1}{=}g^{s_i}\ _{modp}$	First part of the encryption of the receiver notification secret key
$Z_{i2}{=}x_{Bi} * y_A^{s_i}\ _{modp}$	Second part of the encryption of the receiver notification secret key

- Private keys x_i are randomly (or pseudorandomly) generated from $[0, q-1]$
- Public keys are created as: $y_i = g^{x_i} modp$
- Fragments of the message $M[j]$ to be encrypted are in the range $0 < M[j] < p$

The notation used in the description of the protocol is included in Table 1.

B. PHASES OF THE PROTOCOL

The exchange protocol that assures the fairness of the exchange and fulfill the confidentiality requirements listed in § III includes three phases. These phases are called *Creation*, *Acceptation* and *Finalization*. A fourth phase, *Cancellation*, is optional. There is also a *Verification* process to check the status of the exchange. The main flow of the protocol in depicted in Figure 2. In this Figure, the three actors are represented at top (Sender, Blockchain and Receivers) and, as it is illustrated in the outline, all phases are executed *on-chain*. Before the deadline $term_1$, there are depicted the phase 1 (*Creation*), which is executed by the sender, and the phase 2 (*Accept*) executed by all the receivers who agree to get the notification. In the Figure, between the two red lines, which represents the deadlines $term_1$ and $term_2$, there is the *Finish* phase that is performed by the sender for each particular receiver and, at the end of this phase, the receiver is able to decrypt the notification content. If the decryption has not been

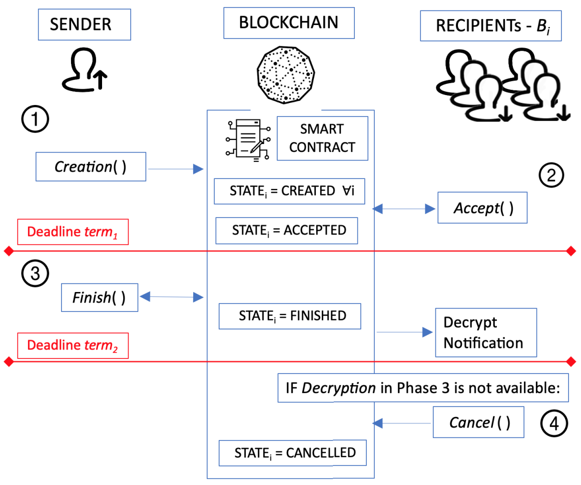

FIGURE 2. Overview of the protocol's phases.

successful, there is an optional phase (*Cancellation*) that can be executed by any receiver to conclude the notification in a fair way. The phases are described in this section.

1) **Creation** Subprotocol 1. A encrypts the message M in cypher text C using a variant of ElGamal encryption and the Discrete Logarithm Integrated Encryption Scheme (DLIES).[2] To do this, the sender A uses a secret encryption element r to generate the final key to encrypt the message M using an XOR cypher operation. If it is necessary, the message can be fragmented in $M[j]$ and, thus, the result of the encryption will be the $C_2[j]$ fragments. Then, C_1 represents the bit commitment with secret encryption element r that will be used by the Smart Contract during the *Finish* phase to verify that all receivers have access to the same r and, therefore they can decrypt the same content of the eDelivery.

[2]Encryption method standardized in ANSI X9.63, IEEE 1363a, and ISO/IEC 18033-2

Subprotocol 1 Step 1. Creation

```
1. $A$ :
    Generation of $M, r, y_A = g^{x_A}_{modp}$
    $key = random.seed(hash(r))$
    Encryption of $M$.
        If required, fragmentation of $M$ in blocks: $M[j]$
        $C_1 = g^r_{modp}$
        FOR $j = 1$ TO $M.length$
            $C_2[j] = M[j]$ XOR $key$
            $key = random.seed(hash(key))$
2. $A \blacktriangleright SM.creation(A, B, C_1, C_2, term_1, term_2,$
   $y_A, g, p, q, D)$
3. $SM$ :
    $State_i = Created, \forall i$
```

To generate a new eDelivery, sender A has his own pair of notification keys created specifically for this exchange, (x_A, y_A). Next, A creates a new instance of a smart contract to manage the new notification delivery by invoking the factory constructor function provided by the service provider, including the following parameters: the encrypted message $\{C_1, C_2\}$, the public notification key y_A, the addresses of the proposed receivers (the set B) and the deadlines $term_1$ and $term_2$. The first one ($term_1$) is the deadline for each receiver to accept the delivery. The second deadline ($term_2$) specifies the valid period for the sender to finish the exchange. Also $term_2$ designates the moment since when the receivers will be able to have evidence of origin and the plain message or, if the protocol run has not finished successfully, they can obtain evidence of the cancellation of the process that has not been properly completed by the sender. At this point, we want to highlight the purpose of the parameter C_1: the parameter represents the bit commitment of sender A with the encryption secret key r, in such a way that by making public this parameter (C_1) the smart contract can publicly verify that the sender is sending the same key to all recipient and, the key is the one that A is being committed to send once C_1 has been made public. We will see how, in our protocol, A sends r in an encrypted envelope created with a shared secret key between each receiver and the sender. These shared keys are created by each recipient and they are confidentially sent to the sender in the *Accept* phase. Then, at the *Finish* phase, the smart contract will check that C_1 contains the right key r using the Schnorr ZKP primitive without the need to know r.

A payment for the service or a deposit D can be included in this stage. Even if fairness is assured in any case, in order to avoid dishonest behavior and fraud attempts, the protocol includes a penalty mechanism to avoid that this behavior can cause bother to other users, in terms of delay or differences on the distribution of execution costs. For this reason we have included in this phase of the protocol a deposit with the aim to encourage the sender to conclude the exchange in the expected way, that is, following the phases of the protocol. As it will be explained in the *Finish* phase the deposit D will be returned to the sender if he finishes the exchange according to the protocol.

2) **Accept** Subprotocol 2. In this multiparty scenario, each receiver can decide individually whether to accept the delivery or not, by executing the corresponding function of the smart contract before the deadline $term_1$. If a receiver accepts the delivery, he executes a function of the smart contract expressing his will. If the receiver B_i does not accept before $term_1$, a rejection is assumed ($State_i = Rejected$), otherwise the delivery has been accepted by the receiver B_i ($State_i = Accepted$). Since A has to allow the access to the delivered contents to those members of B that have accepted the delivery while keeping it confidential for the rest of the world, the plain message cannot be included in a transaction nor stored in the blockchain. Thus, the sender A must generate the necessary elements to confidentially deliver the key to decipher the message, (that is, sending the key that has been committed to send at the *Creation phase*, C_1). The smart contract must ensure that all the receivers that have accepted the delivery can access the same message. In order to achieve this goal, each receiver B_i generates its own pair of shared notification keys x_{Bi}, y_{Bi}. This pair of keys are called *shared* because they will be shared between a particular recipient and the sender. While x_{Bi} is the private key because it is only known by the recipient and the sender, y_{Bi} is the public key because everybody can know it. This public shared notification key will be used in the next phase (*Finish*) by the smart contract to check whether all receivers are able to decrypt the same message and, thus, they can read the same notification content. Accordingly, B_i generates parameters $\{Z_{i_1}, Z_{i_2}\}$ that will allow sender A to get the shared key x_{Bi} as we will see at the *Finish* phase. The calling to the function *accept*() also includes the parameters y_{Bi} and $challenge_i$. The latter is a uniformly random chosen challenge that the sender should appropriately respond in conjunction with the secret encryption element r. The response will be used by the smart contract to verify that every recipient receives the same key r at the *Finish* phase. Note that, in spite of being able to verify the correctness of the secret r, the smart contract will not be able to know the secret r thanks to the use of the ZKP primitive.

Subprotocol 2 Step 2. Accept

```
1. $B_i$ :
   generation of the parameters: $y_{B_i} = g^{x_{B_i}}_{modp}$, $s_i$
   $Z_{i_1} = g^{s_i}_{modp}$, $Z_{i_2} = x_{B_i} * y^{s_i}_{A\,modp}$
2. $B_i$ ► $SM.accept(Z_{i_1}, Z_{i_2}, y_{B_i}, challenge_i)$
3. SM:
   IF($now < term_1$) AND ($Id == B_i$) AND
   ($State_i == Created$)
      $State_i = Accepted$
      Add $B_i$ to $B'$
```

Since each receiver is free to accept the eDelivery or not, he can't be punished if he does not accept the eDelivery. For this reason no penalty mechanism has been included in this phase of the protocol.

3) **Finish** Subprotocol 3. Finally, before the deadline $term_2$, A can finish the delivery process for those B_i who have accepted the notification, executing the *finish()* function of the smart contract.

The sender A generates for each receiver in B', that is, receivers that have accepted the delivery, a response to

Subprotocol 3 Step 3. Finish

```
1. $A$ :
   $x_{Bi} = \left(Z_{i_1}{}^{x_A}\right)^{-1} * Z_{i_2 \, modp}$
   generation of $w_i = r + challenge_i * x_{Bi \, modq}$
2. $A$ ► SM.finish($w_i$)
3. SM:
   IF ($Id == A$) AND (($term_1 < now < term_2$) OR
   (($B' == B$) AND ($now < term2$))
      FOR ($\forall B_i \in B'$)
         IF ($g^{w_i} == g^r * y_{Bi}^{challenge_i}{}_{modp}$)
            $State_i = Finished$
      FOR ($\forall B_i \notin B'$)
         $State_i = Rejected$
      Deposit $D$ is refunded to $A$
4. $B_i$ :
   $r = w_i - challenge_i * x_{Bi \, modq}$
   $key = random.seed(hash(r))$
   FOR $j = 1$ TO $n$
      $M[j] = C_2[j]$ XOR $key$
      $key = random.seed(hash(key))$
```

the challenge in the form of a ZKP using the secret element r used in the encryption of the message in the first step of the protocol, the challenge provided in phase 2 and B_i's secret shared notification key, x_{Bi}. Note that, although the secret shared notification key, x_{Bi}, was created by B_i, B_i has sent the secret shared key to A, encrypted with A's public key using ElGamal cryptosystem, resulting in $\{Z_{i_1}, Z_{i_2}\}$. This way, A can obtain the secret shared notification key for each B_i using its private key: $\left(Z_{i_1}{}^{x_A}\right)^{-1} * Z_{i_2} = x_{Bi}$.

The Smart Contract will store the parameters. With this response, the Smart Contract can verify, by means of the stored data, that each receiver B_i in B' will be able to know the secret element r in order to decipher the message. However, the Smart Contract will not know this element and therefore the message will remain confidential. Thus, the ZKP allows the Smart Contract to verify the commitment with the secret key r that did A in the *Creation* phase of the eDelivery, in such a way that A can proof to the Smart Contract that he is sending the proper secret element r to each receiver without disclosing its value, because his response w_i to the challenge sent by any receiver $challenge_i$ is coherent with the committed value of the secret key, publicly expressed in C_1. Therefore, if $g^{w_i} == g^r * y_{Bi}^{challenge_i}{}_{modp}$ holds, then the Smart Contract can be sure that the receiver is able to get access to the right key and the state of the eDelivery for this receiver will be *Finished*.

In the last step of this phase, receivers can isolate r from w due to knowledge of x_{Bi} and *challenge* and, then, they will be able read the content of the message.

At this point, if the sender has executed *Finish* in the right period of time, and if the verification performed by the smart contract stands, since the sender A has followed the protocol, the smart contract returns the amount D, deposited in the *creation* phase, to the sender A. With this procedure, the protocol wants to avoid a misbehavior from A. If at *term2 Finish* has not been executed satisfactorily, then the deposit D will not be returned. As it will be explained in phase *Cancel*, in this situation, a receiver that has accepted the eDelivery can execute *Cancel* and the exchange will remain in a fair situation. However, this has to be executed after *term2* and it is the receiver who has to execute the function in the smart contract, causing delay and a different distribution of the execution costs. It has to be said that if no receiver accepts the eDelivery then the deposit D is also returned to the sender.

Finally, after *term*2, each receiver in B' (receivers that have accepted the delivery) can access the message through the WEB3 or similar interface or, in the event that the issuer has not successfully completed the protocol, the receivers will have access to the corresponding cancellation evidence.

4) **Cancellation** Subprotocol 4. Cancellation of acceptance. This step is optional and it will be executed by any receiver B_i, if the sender A does not finish the exchange providing the decryption key when the receiver has accepted the contract.
If a receiver tries to cheat executing *Cancel* when the state is not *Accepted* (then it is *finished*), the protocol will not change the state, maintaining fairness. However, the dishonest behavior of this receiver can be punished including him in a blacklist of dishonest users.

Subprotocol 4 Cancellation of Acceptance

```
1. $B_i$ ► SM.cancel()
2. SM:
   IF($now >= term2$, $Id == B_i$ AND
   $State_i == Accepted$)
      $State_i = Cancelled$
```

5) **Verification** The verification process can be carried out by both the sender A and any receiver B_i, as well as any third party involved. In this verification process the status of each receiver can be checked, with the *getState* function, as well as any variables involved in the exchange, as they are public in the smart contract, except the message M, which for reasons of confidentiality, is encrypted.
In addition, as the verification of the ZKP is performed on-chain, in the *Finish* function, we can assure that the state of any receiver B_i is checked by the contract itself.

VI. SMART CONTRACT

The protocol presented in Section § V has been implemented to prove its viability and also to check the execution costs. For this implementation we have used the Ethereum blockchain. This will offer us more functionalities than conventional blockchains such as Bitcoin, using programs called *Smart Contracts*, which allow us to program on a distributed *Turing complete* machine, developed in *Solidity* language, and to store system state changes. Ethereum will use *Ether*, its cryptocurrency, to meter and limit the costs of resources used to execute the code. This section includes the development procedure and the description of the smart contracts.

In this protocol, a *Smart Contract* is used to manage the distribution of a confidential message from a sender A to several recipients (all recipients are the members of a set called B), and to exchange non-repudiation of origin evidence against non-repudiation of reception evidence.

To do this, we need some data structures to keep the group of recipients, thus an array is declared to keep their addresses. Also, a mapping structure has been declared to keep track of the state of each exchange for every recipient, and the variables *z1*, *z2*, *yb*, *c* and *w* (that correspond to the elements Z_{i1}, Z_{i2}, y_{Bi}, $challenge_i$ and w_i according to the notation of the protocol) of the implementation of the ZKP, for each recipient. This mapping allows us to maintain the cast and the time for searching the information from a receiver as constant values. On the other hand, the array lets us to iterate though all the addresses of the receivers.

In addition to that, two variables (*term1* and *term2*) are also declared in the smart contract to store the correspondent deadlines and, also, an additional variable (*sender*) to save the sender's address. We have defined the variable called *start* to set the current time when a delivery is created, and the deadlines *term1* and *term2* are set from this value. A new variable *acceptedReceivers* is used to count the amount of recipients that have accepted the delivery. This way, the sender can finish the delivery of the data and he does not need to wait until *term1*. Thanks to the *acceptedReceivers* variable, the smart contract does not have to verify the state of the exchange from all the receivers, thus we are avoiding an expensive operation in terms of gas consumption thanks to this variable.

We also need to store all the general variables of the zero knowledge proof (ZKP): *c1*, *c2*, *ya*, *g*, *q* and *p* (C_1, C_2, y_A, g, q and p). All data structures are written down in Listing 1.

The following variables are initialized when a new instance of the *Smart Contract* is created: first an array of recipients is initialized with the values sent by the sender, and next the delivery's state for each recipient is set to *created*. We will also store the timeouts and all the general variables for the ZKP and the sender makes a deposit in Ethers.

The *Accept* function verifies that the user who has called the function (i.e. *msg.sender*) has the address in the recipients mapping (see Listing 2), and the correspondent delivery state is *created*. Then, the function also verifies that *term1* deadline has not been overtaken. The delivery state of the recipient will be updated to *accepted* if all verifications are fulfilled, and

```
enum State {notexists, created, cancelled,
    accepted, finished, rejected }

struct ReceiverState{
        bytes z1;
        bytes z2;
        bytes yb;
        bytes c;
        bytes w;
        State state;
}

address public sender;
address[] public receivers;
mapping (address => ReceiverState)
    public receiversState;
uint acceptedReceivers;

bytes public c1;
bytes public c2;
bytes public ya;
bytes public g;
bytes public p;
bytes public q;

uint public term1;
uint public term2;
uint public start;
```

Listing 1. Data structures.

```
function accept(bytes _z1, bytes _z2, bytes _yb,
    bytes _c) public {
  require(now < start+term1, "The␣timeout␣term1␣
    has␣been␣reached");
  require(
    receiversState[msg.sender].state==State.
    created,
    "Only␣receivers␣with␣'created'␣state␣can␣
    accept");

  acceptedReceivers = acceptedReceivers+1;
  receiversState[msg.sender].z1 = _z1;
  receiversState[msg.sender].z2 = _z2;
  receiversState[msg.sender].yb = _yb;
  receiversState[msg.sender].c = _c;
  receiversState[msg.sender].state=State.accepted;
}
```

Listing 2. Accept function.

all variables for the ZKP who depends on each receiver are stored for that particular receiver.

The *Finish* function is depicted in Listing 3. This function fist verifies that the current time is greater than *term1* or whether all recipient have accepted the exchange or not. So as to carry on with the function, it is also necessary to check that the user who has invoked the function is the sender of the eDelivery. Finally, the function also checks that the w variable fulfills the equality $g^w mod\ p = (c1 * (y_b{}^c\ mod\ p))mod\ p$, using the solutions explained in § VI-B. Then, if all of these verifications are met, the delivery states of all the recipients that have been *accepted* are updated to *finished*. Also, for the recipients which state is *created*, the state is changed to *rejected*, and the sender will be refunded.

Lastly, we have depicted the *Cancel* function in Listing 4. This function first verifies that the current time is greater than the deadline *term2* and that the caller of the function is one of the recipients whose delivery state is *accepted*.

```
function finish(address _receiver, bytes _w)
    public {
  require((now >= start+term1) ||
      (acceptedReceivers>=receivers.length),
          "The␣timeout␣term1␣has␣not␣been␣reached␣
     and␣not␣all␣receivers␣have␣been␣accepted␣the␣
     delivery");
  require (msg.sender==sender, "Only␣sender␣of␣the
    ␣delivery␣can␣finish");

  bytes memory check_1=bignumber_modexp(g, _w, p);
  bytes memory check_2=bignumber_modmul( c1,
      bignumber_modexp(receiversState[_receiver].
    yb,
      receiversState[_receiver].c, p), p);

  require (bignumber_equals(check_1, check_2),
          "(g^w␣mod␣p)␣and␣(((g^r␣mod␣p)(ybc␣mod␣p
  ))␣mod␣p)␣are␣not␣equals");
  sender.transfer(this.balance);
  for (uint i = 0; i<receivers.length; i++) {
    receiversState[receivers[i]].w = _w;
    if (receiversState[receivers[i]].state ==
        State.accepted) {
      receiversState[receivers[i]].state =
        State.finished;
    } else if (receiversState[receivers[i]].state
    ==
        State.created) {
      receiversState[receivers[i]].state =
        State.rejected;
    }
  }
}
```

Listing 3. Finish function.

```
function cancel() public {
  require(now >= start+term2,
      "The␣timeout␣term2␣has␣not␣been␣reached");
  require(receiversState[msg.sender].state==State.
    accepted,
      "Only␣receivers␣with␣'accepted'␣state␣can␣
    cancel");

  receiversState[msg.sender].state = State.
    cancelled;
}
```

Listing 4. Cancel function.

If these conditions are met, then the smart contract updates the receiver's delivery state to *cancelled*.

A. FACTORY CONTRACT

We have also used the well-known factory method programming pattern in order to generate new instances of a smart contract to manage each new registered eDelivery. We have created the *Factory Contract* that generates and deploys new contracts for this purpose, the use of this pattern will achieve the following advantages:

- The code used to generate and manage a new registered eDelivery will be reusable.
- The factory also works as a storage of the addresses of the child contracts, the eDeliveries.
- We can easily access all the eDeliveries that we have made using this service thanks to this new address storage inside the factory contract.
- The provider of the central service only has to pay for the deployment of the factory contract.
- Each user who wants to send a new eDelivery has to pay for the creation of a new instance. We have analyzed the deployment costs in Section § VIII. According to this pattern, the factory contract is in charge of the deployment of new instances to manage a new eDelivery and it will keep the address of this new deployed contract in an inner data structure.

B. ZKP IN SOLIDITY

Implementing the ZKP in solidity is challenging due to the use of big numbers. An inefficient implementation would result in high execution costs in terms of gas.

The most important problem found to implement the zero knowledge proof in Solidity is that we need to operate with numbers of minimum 256 bits, and there is an operation that checks an equality, $g^w mod\ p = (c1 * ({y_b}^c\ mod\ p)) mod\ p$, that needs more bits to operate.

The solution found consists of using the *bytes* data type, that can hold a sequence of bytes of any size, instead of using the *uint* data type, that can hold unsigned integers of maximum 256 bits. If we use this data type in the parameters of the functions and to store values, we can use this Smart Contracts with numbers of any size, like 256, 512, 1024 or 2048 bits.

Then, we have also used the Big Number Library for Solidity [3] to operate with this numbers. First of all we convert the *bytes* values to BigNumber values, and then we perform the *equals()*, *modmul()* and *modexp()* functions using this type (see Listing 5).

VII. EVALUATION OF PROPERTIES

Now, this section presents a survey and a security discussion of the desired properties for registered *eDelivery* schemes that were enumerated in Section II.

Regarding the security properties defined in Section II, we have removed the *efficiency* property because it will be evaluated separately, in Section VIII, where the results and a set of experiments to evaluate the performance of the protocol are presented. We also have grouped together the properties of *fairness* and *evidence transferability* for discussion purposes.

1) **Effectiveness.** The system for registered eDelivery presented in this paper is effective. So, all parties will receive the expected items in case of behaving according to protocol.
   To create a new *eDelivery*, the sender generates a new instance of the smart contract to execute the functions following the specifications of the protocol. If all the parties execute all the functions correctly (*Accept()* and *Finish()* functions), they will have the items that they expected to receive. This can easily be deduced from the solidity code depicted in Section § VI. At the end of the exchange, the receivers that have followed the

[3](https://github.com/zcoinofficial/solidity-BigNumber/)

```
function bignumber_equals(bytes _a, bytes _b)
    internal view returns(bool) {
  BigNumber.instance memory a;
  BigNumber.instance memory b;

  a.val = _a;
  a.bitlen = BigNumber.get_bit_length(_a);
  a.neg = false;
  b.val = _b;
  b.bitlen = BigNumber.get_bit_length(_b);
  b.neg = false;

  // BigNumber.cmp() returns -1 on a<b, 0 on a==b,
  // 1 on a>b
  return BigNumber.cmp(a, b, false)==0;
}

function bignumber_modmul(bytes _a, bytes _b,
    bytes _m) internal view returns(bytes) {
  BigNumber.instance memory a;
  BigNumber.instance memory b;
  BigNumber.instance memory m;

  a.val = _a;
  a.bitlen = BigNumber.get_bit_length(_a);
  a.neg = false;
  b.val = _b;
  b.bitlen = BigNumber.get_bit_length(_b);
  b.neg = false;
  m.val = _m;
  m.bitlen = BigNumber.get_bit_length(_m);
  m.neg = false;

  BigNumber.instance memory res = a.modmul(b, m);
  return (res.val);
}

function bignumber_modexp(bytes _b, bytes _e,
    bytes _m) internal view returns(bytes) {
  BigNumber.instance memory b;
  BigNumber.instance memory e;
  BigNumber.instance memory m;

  b.val = _b;
  b.bitlen = BigNumber.get_bit_length(_b);
  b.neg = false;
  e.val = _e;
  e.bitlen = BigNumber.get_bit_length(_e);
  e.neg = false;
  m.val = _m;
  m.bitlen = BigNumber.get_bit_length(_m);
  m.neg = false;

  BigNumber.instance memory res =
      b.prepare_modexp(e, m);
  return (res.val);
}
```

Listing 5. BigNumber functions.

protocol will have the key to decrypt the delivered data and the non-repudiation of origin proof whereas the sender will have the non-repudiation of rejection proof and the state of this delivery will be *Finished* for every recipient.

2) **Fairness** and **Transferability of evidence.** The proposed protocol for registered *eDelivery* is fair. At the end of a protocol execution, either each party has received the proper element or neither party has received any useful data about the other's element, providing *strong fairness* [1]. Moreover, the evidence generated by the protocol can be transferred to an external party in order to prove the outcome and the effects of the exchange without no further verifications. On the one hand, according to the protocol, the sender is not going to receive the non-repudiation evidence of reception generated by the smart contract except if he executes a transaction in order to allow the receivers to decrypt the message and, at the same time, allows the smart contract to check the validity of the provided key (when *state* = *Finished*). On the other hand, any receiver B_i is able to get access to the delivered data only if he runs a transaction in order to agree to receive the *eDelivery* (case *state* = *accepted*).

If the parties do not follow the specifications of the protocol, that is, if they do not execute the functions *Accept()* and/or *Finish()*, the smart contract guaranties a result that is fair for every user without the need of any intervention of a *TTP*.

We have analyzed all the cases where the protocol can lead the exchange to prove the strong fairness of the protocol.We have used a state transition diagram. For each receiver, this diagram is formed by three final states: *Finished*, *Cancelled* and *Rejected* and two intermediate states: *Created* and *Accepted*. These states cannot be final states because the protocol will eventually change the state in any case. The states and the transactions are represented in Figure 3.

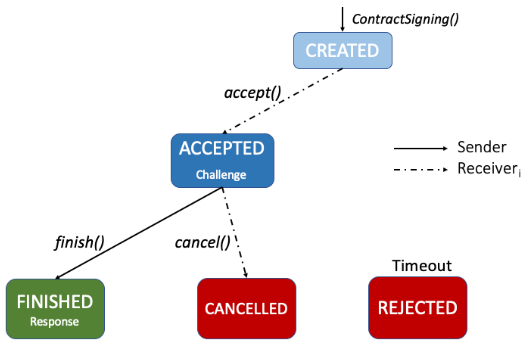

Figure 3. State's life cycle in the Confidential eDelivery Scheme.

Finished implies that the exchange has been completed following all the stages of the protocol, while *Canceled* and *Rejected* represent exchanges that have not been completed, for different reasons. Now, we will show how the protocol leads the exchange to a fair situation in all cases:

- *creation() not executed.* If the first step of the protocol is not executed satisfactorily, the the eDelivery is not created, any element of the fair exchange has been provided and the parties are in a fair situation.
- *accept() not executed satisfactorily.* If, after the creation of the eDelivery in the first step of the protocol, a receiver does not execute *accept*, before the deadline *term1*, then the smart contract will set the state for this receiver to *Rejected*. This way,

after the first timeout, those receivers that have not called the *accept()* function are not included in the set B' and the sender will not finish the exchange with this receivers, so the delivery will not be completed for them and they will not receive the decryption key. These receivers do not have access to the delivered data and to the non repudiation of origin element. In the other hand, the sender do not have a non repudiation of reception proof generated by these receivers. So, the situation is fair for all the parties. The final state for these exchange will be *Rejected*.

- *finish() executed successfully.* The sender can execute *finish()* only for those receivers that have executed *accept()*, taht is, the receivers in B', with state==*accepted*. The sender can, consequently have the non-repudiation of reception proof. When the sender executes *finish()* then the smart contract checks the provided elements and verifies that the receiver will have access to the delivered data and also that he obtains the non-repudiation of origin proof (state==*finished*). This way, after a successful execution of *finish()*, the protocol leads the exchange to a fair state, where the parties have received all the desired elements.
- *finish() not executed successfully. Execution of cancel().* If after the execution of *accept()*, a receiver does not receive the decryption key (that is, the sender has not executed *finish()* or the smart contract does not verify the provided elements), then, after the expiration of the deadline, the receiver can call the *cancel()* function to conclude the exchange with fairness. This way the smart contract will set the state to *Cancelled*, meaning that the exchange has not been performed successfully, that is, the receiver hasn't had access to the delivered data and the protocol has not generated non-repudiation proofs.

According to the previous evaluation, the fairness provided by the protocol is *strong fairness*. Although the smart contract can create finalization and alternative cancellation proofs, the protocol does not allow any circumstance where a user could get contradictory evidence since the state for each receiver is only updated by the smart contract. Therefore, it is not possible to do an action that could lead to an unfair situation (the contrary would be considered *weak fairness* [1]).

The proofs generated during the execution of a protocol run can be submitted as proofs or evidence to an external arbiter. We have to take into account that users cannot get contradictory proofs and valid evidence are only created by the execution sequence of the smart contract. The evidence can be evaluated by an external arbiter who can determine whether the exchange has concluded successfully or not. In addition to that, its transferability is easy, because the proofs generated during the exchange are all stored on the blockchain. Therefore, since the blockchain is immutable, it is not possible to change the content of any message and, thus, the scheme provides message integrity. It is also possible to deduce the exact point in time when the delivery took place from the timestamp of the block where the transaction was included.

3) **Temporal parameters: Timeliness and Timestamping.** A successful delivery will always be completed before deadline *term2*.
If the delivery is not successful we have different deadlines in function of how the exchange has been performed. If a receiver does not accept the delivery, then the delivery will be classified as *Rejected* at *term1*. If after the acceptance of the delivery by a receiver the sender does not finish the exchange, then the exchange can be canceled at *term2*.
Moreover, the system prompts the sender to end an eDelivery exchange before deadline *term2*. This motivation is performed by means of a deposit that is blocked in the smart contract. The deposit will be given back to the sender in case of conclusion before *term2*. We have to take into account that the blockchain timestamps all transactions performed on it.
4) **Non-repudiation.** The *e-Delivery* protocol must provide a Non-Repudiation of reception evidence and a Non-repudiation of origin evidence too.
   - Regarding the origin, a sender A of the *eDelivery* cannot deny having executed the function to create the *eDelivery* since there is a transaction on the blockchain from her address containing the addresses of the receivers and the encrypted message (see Subprotocol 1). In addition, the transaction related with the execution of the *finish(()* function (see Subprotocol 3) proves that the sender has provided the key to decrypt the message to the receivers that accepted the *eDelivery*, and is considered the Non-repudiation of origin element since this transaction leads the exchange to the *Finished* state.
   - Concerning the reception, each recipient B_i is not able to refuse having accepted the eDelivery, because an accepting transaction from his address is stored on the blockchain. This transaction accepts the reception of the eDelivery and, according to that, the smart contract changed the *state* of the eDelivery to *Finished* after the execution of *finish()* function. The protocol assures that each receiver has obtained the right decryption key since the smart contract validates it by using the ZKP (see Subprotocol 2 and Subprotocol 3).
5) **Trusted Third parties.** The proposed scheme does not need the intervention of an external Trusted Third Party. In this protocol, the parties run the functions of the smart contract by generating the appropriate

transactions and, thanks to the use of the smart contract, no further dispute resolution phase is needed. The exchange of information in the proposed system is all on-chain, as a consequence, all communications related to an eDelivery are stored on the blockchain. We have to take into account that the blockchain technology has been conceived to be immutable and publicly verifiable, thus the actions that have been carried out in a protocol run are publicly verifiable and anyone can know which address is accountable for that.

6) **Confidentiality**. A delivery will be confidential if only the receivers that accept the delivery can access the delivered data. For this reason the data cannot be included in clear in any transaction and must not be registered in the blockchain. The data cannot be a parameter in the smart contract functions and the smart contract cannot gain access to the decryption key. Even tough, the protocol must check that the data received by all the receivers is the same.
In Subprotocol 1 the sender runs the *creation* function of the smart contract including C_1 and C_2, that represent the encrypted message. In Subprotocol 2 each receiver provides a way for the sender to send privately the key to decrypt the message, through the elements Z_1 and Z_2. The smart contract verifies in Subprotocol 3 that the right decryption key is provided and that can be derived from w_i, isolating r.
There is no more entities involved in the eDelivery exchange, and the transactions only include encrypted messages and hidden decryption keys that are only recoverable for receivers who have accepted the exchange. Thus, the delivered data will remain confidential.

VIII. PERFORMANCE ANALYSIS

Once we have finished the implementation of the proposal and we have tested the results, we have performed some experiments to determine the efficiency of the system in terms of cost, since the economical execution costs could be a concern in the development of the *eDelivery* service.

The performance analysis includes several tests, performed using the *Smart Contracts* explained in § VI. We have deployed it to the Ganache network, a personal blockchain used for Ethereum development, to isolate the performance conditions and possible problems of a real network like the main Ethereum or the Rinkeby test networks.

From the results of the tests, we have detailed the gas cost of the functions, executing the protocol with different values for the number of receivers, selecting the values of one, two and ten receivers. With this tests we want to obtain absolute values of the economical costs to evaluate the viability of the protocol and also evaluate the performance of multiparty protocols compared to Two-party protocols and for this reason the cost per receiver has been taken into account.

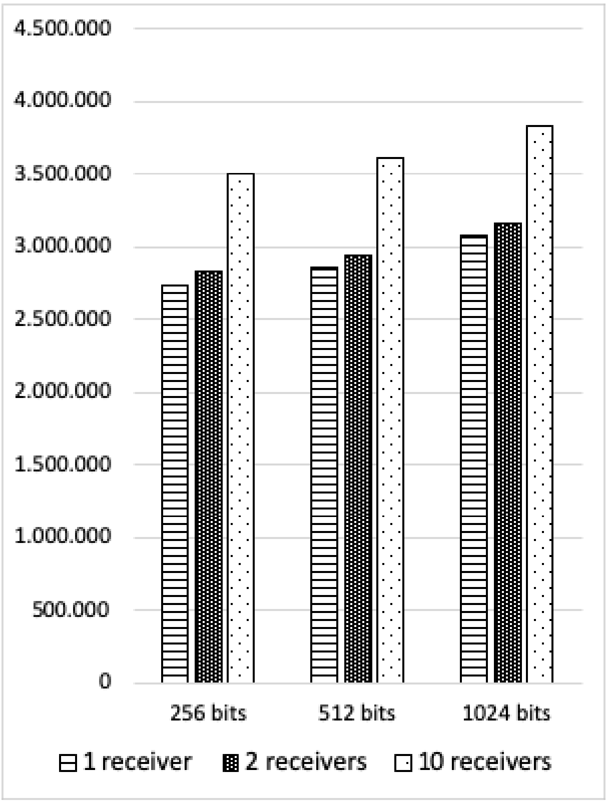

Figure 4. createDelivery() function gas cost of the confidential multiparty protocol.

We have also tested the performance in terms of delay, to have an illustrative reference of the delays introduced by the functions, although we have not included the complete list of results in this paper. In the Ganache network all the functions have been executed with a delay between 55 and 837 milliseconds, closely related to the gas cost of each function. But in a normal production Ethereum blockchain network, all the transactions need a few seconds to be validated through the mining process, which have a delay of 15-30 seconds, being the greatest component of the delay value.

In Figure 4, Figure 5 and Figure 6 we can see the gas cost of the main functions of the protocol. The cost of this analysis is computed in gas, but to know the exact price of this transactions we also need the gas price, set in Ethers, and the Ethers to US-Dollars exchange rate, but neither one nor the other are fixed. For this reason, we have added in Table 2, Table 3 and Table 4 the price in US-Dollars of each of the functionalities as a guideline, considering the exchange rate at January 1, 2020,[4] and taking into account a gas price of 1 Gwei and 20 Gwei. We haven't included the cost in gas of the *deploy()* function of the *factory*, because is independent of the number of receivers and the number of bits: 3.864.642 (between 0.49$ and 9.78$). The main difference, in addition to the total cost of the transaction, is the time it will take to the transaction to be accepted by a mining node. Thus,

[4]Ether Price = 130,27$

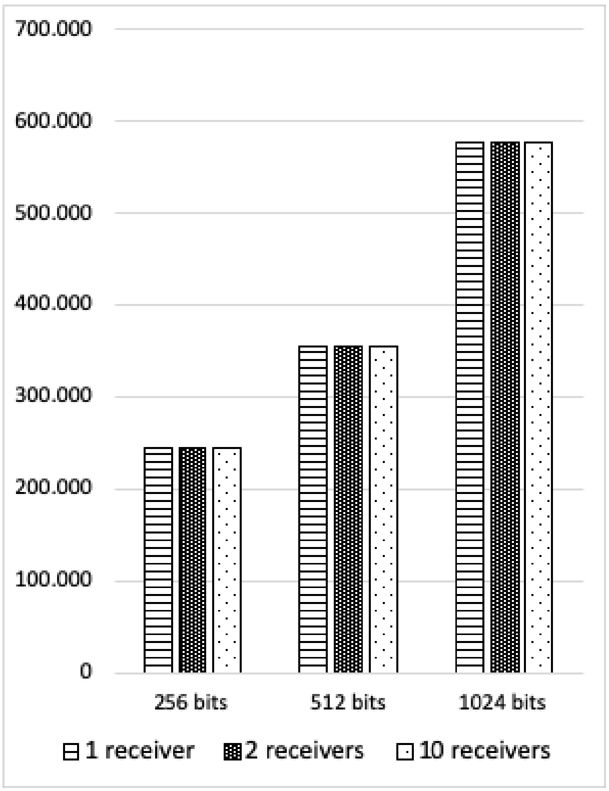

Figure 5. Accept() function gas cost of the confidential multiparty protocol.

TABLE 2. createDelivery() function cost in gas, equivalent US-Dollars price in the interval (1 Gwei - 20 Gwei) gas price, and cost per receiver in the same interval.

Cost in Gas	256 bits	512 bits	1024 bits
1 receiver	2,743,000 (0.35-7.09)	2,826,871 (0.35-7.21)	3,077,925 (0.38-7.78)
2 receivers	2,826,871 (0.35-7.14)	2,938,640 (0.36-7.43)	3,161,732 (0.40-8.00)
Cost per Receiver	(0.17-3.57)	(0.18-3.71)	(0.20-4.01)
10 receivers	3,498,345 (0.43-8.85)	3,610,115 (0.45-9.13)	3,833,208 (0.48-9.70)
Cost per Receiver	(0.04-0.87)	(0.04-0.91)	(0.04-0.96)

TABLE 3. Accept() function cost in gas and equivalent US-Dollars price with 1 Gwei - 20 Gwei gas price.

Cost in Gas	256 bits	512 bits	1024 bits
1 receiver	244,207 (0.03-0.61)	355,403 (0.04-0.90)	577,988 (0.07-1.46)
2 receiver	244,207 (0.03-0.61)	355,403 (0.04-0.90)	577,988 (0.07-1.46)
10 receiver	244,207 (0.03-0.61)	355,403 (0.04-0.90)	577,988 (0.07-1.46)

a transaction made on the main Ethereum network with a value of 1 Gwei can take almost 100 minutes (depending on the current network traffic), while in the case of 20 Gwei can be reduced to 30 seconds.[5]

[5]https://ethgasstation.info/

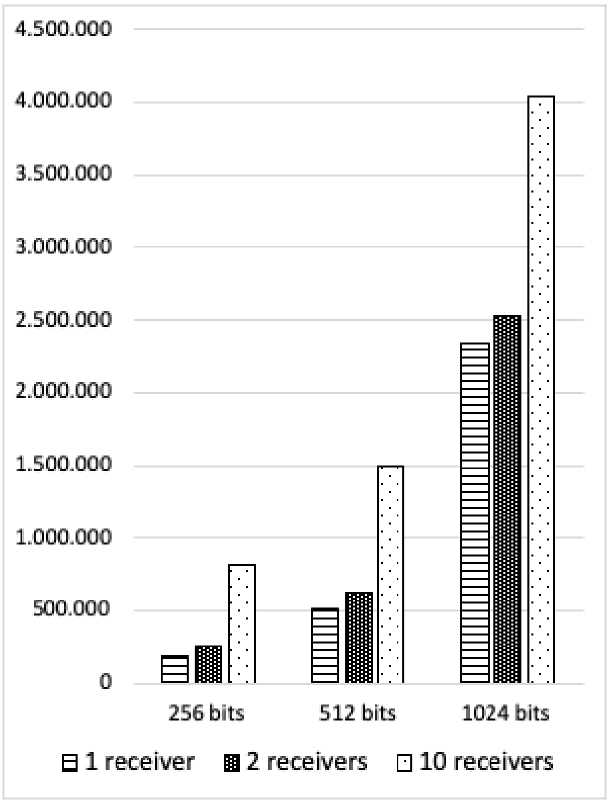

Figure 6. Finish() function gas cost of the confidential multiparty protocol.

TABLE 4. Finish() function cost in gas and equivalent US-Dollars price in the interval (1 Gwei - 20 Gwei) gas price, and cost per receiver in the same interval.

Cost in Gas	256 bits	512 bits	1024 bits
1 receiver	192.681 (0.02-0.49)	516.563 (0.06-1.30)	2.343.311 (0.29-5.92)
2 receivers	261.149 (0.03-0.65)	625.171 (0.07-1.57)	2.532.199 (0.32-6.40)
Cost per Receiver	(0.01-0.32)	(0.03-0.70)	(0.15-3.20)
10 receivers	808.893 (0.10-2.04)	1.494.035 (0.19-3.78)	4.043.303 (0.50-10.22)
Cost per Receiver	(0.008-0.20)	(0.01-0.37)	(0.04-1.01)

From the analysis of the results included in Figure 4, Figure 5, Figure 6, Table 2, Table 3 and Table 4 we can conclude that:

- Cost of the functions:
  - The *factory deploy()* and the *createDelivery()* functions are considerably more expensive than the *accept()* and *finish()* functions. An exception to this statement occurs when we use long keys and the number of receivers is big.
  - The *factory deploy()* is executed only once and its cost is amortized by the number of deliveries created using the factory. We have to take into account that the cost of the two more expensive operations (*factory deploy* and *createDelivery* functions) is

distributed between the owner of the service (the *Factory* deployer) and the sender of the delivery.
  - The only function executed by the receiver, *accept()* is very cheap in terms of gas.
- Number of receivers.
  - The *factory deploy()* and the *accept()* functions are independent of the number of receivers.
  - The *createDelivery()* and the *finish()* functions depend of the number of receivers and the total cost grows as the number of receivers is increased. However, although the total cost grows, the cost per receiver decreases, making the multiparty protocol more efficient with a bigger number of receivers.
- Length of the security elements.
  - The most costly operation, *factory deploy()* function, is independent of the length of the parameters, allowing us to use longer parameters.
  - The cost of execution of the *createDelivery()* and the *accept()* functions rises slightly with the length of the parameters.
  - *finish()* is the function that includes the verification of the ZKP and is the function more affected by the change in the length of the parameters. However, as the number of receivers increases, the cost introduced by the use of longer parameters is reduced.
- Execution delay
  - The execution delay is not greater than 900 ms in any case. This means that the execution time is small compared with the block validation time in the Ethereum network through the mining process, so this will be the greatest component of the total delay value.
  - The validation delays are closely related with the gas cost used in each function. This way a user can choose the gas cost values, analyzed above, that best adjust to its cost and delay requirements That is, the delay and the cost of the eDelivery can be controlled adjusting the gas price.
  - As it could be expected, the greatest execution time correspond to the *finish()* function, where the ZKP is checked while *accept()* is the quickest function. In addition, longer parameters require bigger execution times.

IX. COMPARISON WITH PREVIOUS STUDIES

It has been proved that blockchain-based technologies can be very useful in the design of a qualified registered eDelivery service, solving the problems related with the use of TTPs.

The results of the analysis of properties included in VII prove that the protocol achieves the ideal properties of the service: effectiveness, fairness, timeliness, non-repudiation and confidentiality, as it is summarized in a table that compares the new protocol with previous studies Table 5.

In [20] we proposed two protocols to send multiparty eDelivery notifications also based on blockchain technology.

TABLE 5. Comparison of Properties.

Property	Non-Confidential Protocol [20]	Confidential Protocol With TTP [20]	Confidential Protocol Without TTP
Effectiveness	YES	YES	YES
Fairness	STRONG	WEAK	STRONG
Timeliness	YES	YES	YES
Non-repudiation	YES	YES	YES
Confidentiality	NO	YES	YES
Evidence Transferibility	YES	NO	YES
Use of TTP	NO	OPTIMISTIC	NO

However, with respect to the scheme we propose here, the first protocol in [20] does not have the property of confidentiality, since the content of the notification is public. Although the second eDelivery proposal in [20] achieves confidentiality, in order to assure the fairness of the exchange, it needs the intervention of an independent trusted third party (TTP) that takes actions as an intermediary to resolve disputes between sender and receivers. Thus, the proposed scheme in this paper improves the previous systems because it achieves confidentiality with respect to the former proposal in [20], and it removes the intervention of the TTP with respect to the latter proposal in [20].

In addition to that, our new scheme not only provides fairness and confidentiality without needing the intervention of a TTP thanks to the implementation of a ZKP primitive inside the protocol, but also the fairness that guarantees is *strong*. In [20] the protocol that has confidentiality has some cases where users (sender and receivers) can have contradictory evidence. That is, for example, a malicious sender can obtain, at the same time, a non-repudiation evidence from a receiver and a cancellation evidence from the smart contract. Thus, the sender can get evidence that a certain notification has been cancelled or has been finished successfully. That is the reason why the fairness of the previous proposal is *weak*, in contrast to the *strong* fairness of the new proposal of this paper.

Summarizing the comparison of features included in Table 5 we can say that the presented protocol not only allows the protocol to be executed without the involvement of a TTP but also improves the behavior of the previous confidential protocol offering strong fairness and transferability of evidence. These properties were only achieved by a non-confidential protocol.

Regarding the performance of the proposals, we can say that the confidential protocol without TTP executes more on-chain operations that the confidential protocol with TTP. For this reason the gas cost could be bigger. However, we have to take into account that this proposal does not require the payment for the services of the TTP, so the gas cost has to be added to the cost of the TTP for the previous protocol. Moreover, the new protocol obtains greater benefits of the multiparty operation, since the previous protocol executes independent resolutions of the exchange for each receiver.

Table 6 shows the differences on the cost per receiver between the confidential protocol with TTP and the new

TABLE 6. Finish() function cost per recevier in US-Dollars in the interval (1 Gwei - 20 Gwei) gas price.

Cost per receiver	Without TTP 256 bits	With TTP
1 receiver	(0.02-0.49)	(0.019-0.39)+TTP cost
2 receivers	(0.01-0.32)	(0.019-0.39)+TTP cost
10 receivers	(0.008-0.2)	(0.019-0.39)+TTP cost

confidential protocol without TTP, for the function that differs the most between the two protocols, *finish()* function. In this comparison we have to take into account two important facts. The first one in that in the protocol with TTP the *finish()* function is not mandatory. The second one is that while in the protocol without TTP the *finish()* function is executed once, independently of the number of receivers, in the protocol with TTP the function is executed once for each receiver. Thus, the cost for the protocol without TTP decreases with the number of receivers.

To conclude this comparison we have evaluated the delay in the execution of the functions in both protocols. For the confidential protocol without TTP, after the execution of the functions, the block including the transaction has to be confirmed in the blockchain. The execution time for all the functions is lower than 900 ms while the block confirmation delay depends on the gas price but can be included in the interval of 15-30 seconds, being the main component of the delay of the protocol. This delay can be considered much lower than the delay introduced by the resolution of an exchange executed by a TTP. When a user asks the TTP for a resolution, the TTP has to queue the request, check the received data and make a decision about the final result of the exchange, so the user could have to wait a considerable amount of time to obtain a response, compared with the block confirmation time.

X. CONCLUSIONS AND FUTURE WORK

The paper proposes a new protocol that achieves the fulfillment of all the desired properties of a registered eDeliveries service using blockchain. Since now, these properties were partially achieved by the existing protocols. So the new scheme includes the best part of each one of the previous protocols in a single protocol. This way, the new protocol allows the eDelivery of confidential data without the need of a trusted third party. The new protocol has been detailed, implemented and analyzed, obtaining the following conclusions:

- The presented protocol uses blockchain to allow a fair exchange for registered deliveries. This way, data, proof of origin and proof of reception are exchanged without the involvement of any trusted third party while achieving the desired properties for fair exchange protocols: efficiency, strong fairness, transferability, non-repudiation and confidentiality. For this reason, the protocol is an obvious improvement of previous proposals.
- The protocol allows multiparty deliveries, useful to reduce costs in those scenarios in where the same delivery involve several receivers, like notifications for enterprise staff, shareholders meetings,...
- The delivered data is confidential, only sender and receiver can access the data. This means that the smart contract must manage the exchange and assure fairness without being able to access the delivered data. In protocol terms this means that the delivered data must be encrypted until acceptance, when the non-repudiation of reception is provided, and the smart contract, without having access to the decryption key, must assure that all the receivers will be able to decrypt the same message.
- To achieve all the desired properties, including confidentiality, the protocol uses more complex cryptography than previous protocols. The protocol has been implemented and tested on Ethereum to analyze the performance of these operations. The results show that although the execution costs are slightly greater than those of non-confidential protocols, the protocol is viable. The smart contracts are described in the paper.
- With the introduction of this new protocol, users can choose between protocols that register the contents of the delivered data or the confidential protocol. Due to execution costs, users must use the non-confidential protocol not only when the data registry in the blockchain is required, but also in all the deliveries where confidentiality is not required.
- The performance analysis shows that the execution costs depend on the number of receivers, the gas price and the length of the cryptographic parameters. The efficiency of the protocol rises with the number of receivers. The gas price will affect the execution delay, but since the deliveries are asynchronous the execution delay is not a critical parameter. We have also evaluated the execution delay but the resulting values are small enough compared to the block verification time, so the total delay depends strongly of the block verification time. The length of the cryptographic parameters are related with the strength of the confidentiality but they don't have any effect on the strength of the fairness offered by the protocol.

The performance analysis has allowed the detection of the most costly operations. The use of longer parameters leads to higher levels of protection but are expensive in terms of cost. As a future work we intend to implement the protocol using elliptic curve cryptography. According to our initial tests, we will expect to have similar results to those obtained with lengths of 256 bits while offering higher security standards. Thus, the Table 6 could have the appropriate parameters to establish a comparison point for future works.

REFERENCES

[1] N. Asokan, V. Shoup, and M. Waidner, "Asynchronous protocols for optimistic fair exchange," in *Proc. IEEE Symp. Secur. Privacy*, Oakland, CA, USA, May 1998, pp. 86–99.

[2] K. Biswas and V. Muthukkumarasamy, "Securing smart cities using blockchain technology," in *Proc. IEEE 18th Int. Conf. High Perform. Comput. Commun., IEEE 14th Int. Conf. Smart City, IEEE 2nd Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS)*, Dec. 2016, pp. 1392–1393.

[3] S. Delgado-Segura, C. Pérez-Solà, G. Navarro-Arribas, and J. Herrera-Joancomartí, "A fair protocol for data trading based on bitcoin transactions," *Future Gener. Comput. Syst.*, vol. 107, pp. 832–840, Jun. 2020.

[4] *Electronic Signatures and Infrastructures (ESI); Registered Electronic Mail (REM) Services*, document ETSI EN 319 532-5.

[5] *Security Guidelines on the Appropriate Use of Qualified Electronic Registered Delivery Services. Guidance for Users, Version 2.0*, Eur. Union Agency Netw. Inf. Secur., Heraklion, Greece, Dec. 2016.

[6] E. Heilman, L. AlShenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg, "TumbleBit: An untrusted bitcoin-compatible anonymous payment hub," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, San Diego, CA, USA, 2017.

[7] S. Goldfeder, J. Bonneau, R. Gennaro, and A. Narayanan, "Escrow protocols for cryptocurrencies: How to buy physical goods using bitcoin," in *Proc. 21st Int. Conf., Financial Cryptogr. Data Secur. (FC)*, in Lecture Notes in Computer Science, vol. 10322, 2017, pp. 321–339.

[8] H. R. Hasan and K. Salah, "Blockchain-based solution for proof of delivery of physical assets," in *Blockchain—ICBC* (Lecture Notes in Computer Science), vol. 10974. Cham, Switzerland: Springer, Jun. 2018, pp. 139–152.

[9] H. R. Hasan and K. Salah, "Proof of delivery of digital assets using blockchain and smart contracts," *IEEE Access*, vol. 6, pp. 65439–65448, 2018.

[10] E. Heilman, F. Baldimtsi, and S. Goldberg, "Blindly signed contracts: Anonymous on-blockchain and off-blockchain bitcoin transactions," in *Financial Cryptography and Data Security*. Berlin, Germany: Springer, 2016, pp. 43–60.

[11] Q. Huang, G. Yang, D. S. Wong, and W. Susilo, "A new efficient optimistic fair exchange protocol without random oracles," *Int. J. Inf. Secur.*, vol. 11, no. 1, pp. 53–63, Feb. 2012.

[12] Q. Huang, D. Wong, and W. Susilo, "P^2OFE: Privacy-preserving optimistic fair exchange of digital signatures," in *Topics in Cryptology—CT-RSA*. Cham, Switzerland: Springer, 2014, pp. 367–384.

[13] H. Kılınç and A. Küpçü, "Optimally efficient multi-party fair exchange and fair secure multi-party computation," in *Topics in Cryptology—CT-RSA* (Lecture Notes in Computer Science), vol. 9048. Cham, Switzerland: Springer, Mar. 2015, pp. 330–349.

[14] S. Kremer and O. Markowitch, "A multi-party non-repudiation protocol," in *Proc. 15th Int. Conf. Inf. Secur. IFIP World Comput. Congr.*, 2000, pp. 271–280.

[15] J. Liu, W. Li, G. O. Karame, and N. Asokan, "Toward fairness of cryptocurrency payments," *IEEE Secur. Privacy*, vol. 16, no. 3, pp. 81–89, May/Jun. 2017.

[16] M. Mut-Puigserver, M. M. Payeras-Capellà, and M. A. Cabot-Nadal, "Blockchain-based fair certified notifications," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology* (Lecture Notes in Computer Science), vol. 11025. Cham, Switzerland: Springer, 2018, pp. 20–37.

[17] M. Mut-Puigserver, M. M. Payeras-Capella, and M. A. Cabot-Nadal, "Blockchain-based contract signing protocol for confidential contracts," in *Proc. IEEE/ACS 16th Int. Conf. Comput. Syst. Appl. (AICCSA)*, Nov. 2019, pp. 1–6.

[18] J. A. Onieva, J. Zhou, and J. Lopez, "Multiparty nonrepudiation: A survey," *ACM Comput. Surveys*, vol. 41, no. 1, pp. 5:1–5:43, Jan. 2009.

[19] M. Payeras-Capella, M. Mut-Puigserver, and M. A. Cabot-Nadal, "Smart contract for multiparty fair certified notifications," in *Proc. 6th Int. Symp. Comput. Netw. Workshops (CANDARW)*, Takayama, Japan, Nov. 2018, pp. 459–465.

[20] M. M. Payeras-Capella, M. Mut-Puigserver, and M. A. Cabot-Nadal, "Blockchain-based system for multiparty electronic registered delivery services," *IEEE Access*, vol. 7, pp. 95825–95843, 2019, doi: 10.1109/ACCESS.2019.2929101.

[21] Z. Shao, "Fair exchange protocol of Schnorr signatures with semi-trusted adjudicator," *Comput. Elect. Eng.*, vol. 36, no. 6, pp. 1035–1045, 2010.

[22] M. Swan, *Blockchain: Blueprint for a New Economy*. Newton, MA, USA: O'Reilly Media, 2015. [Online]. Available: http://shop.oreilly.com/product/0636920037040.do

[23] *Regulation (EU) No 910/2014 of the European Parliament and of the Council of 23 July 2014 on Electronic Identification and Trust Services for Electronic Transactions in the Internal Market and Repealing. Directive 1999/93/EC*, The European Parliament and the Council of the Europena Union, Brussels, Belgium, 1999.

[24] J. Zhou, R. Deng, and F. Bao, "Some remarks on a fair exchange protocol," in *Proc. 3rd Int. Workshop Pract. Theory Public Key Cryptosyst. (PKC)*, in Lecture Notes in Computer Science, vol. 1751. New York, NY, USA: Springer-Verlag, Jan. 2000, pp. 46–57.

[25] *Transitions: Recommendation for the Transitioning of the Use of Cryptographic Algorithms and Key Lengths*, DRAFT NIST Special Publication 800-131A, Mar. 2019, doi: 10.6028/NIST.SP.800-131Ar2.

[26] *Digital Signature Standard (DSS). (U.S. Department of Commerce, Washington, D.C.), Federal Information Processing Standards Publication (FIPS) 186-4*, NIST, Gaithersburg, MD, USA, Jul. 2013, doi: 10.6028/NIST.FIPS.186-4.

[27] *Secure Hash Standard (SHS). (U.S. Department of Commerce, Washington, D.C.), Federal Information Processing Standards Publication (FIPS) 180-4*, NIST, Gaithersburg, MD, USA, Aug. 2015, doi: 10.6028/NIST.FIPS.180-4.

[28] SECG. (2010). *Standards for Efficient Cryptography (SEC) 2: Recommended Elliptic Curve Domain Parameters. Version 2.0*. [Online]. Available: https://www.secg.org

[29] C. Esposito, F. Palmieri, and K.-K. R. Choo, "Cloud message queueing and notification: Challenges and opportunities," *IEEE Cloud Comput.*, vol. 5, no. 2, pp. 11–16, Mar./Apr. 2018, doi: 10.1109/MCC.2018.022171662.

[30] N. Zupan, K. Zhang, and H.-A. Jacobsen, "Hyperpubsub: A decentralized, permissioned, publish/subscribe service using blockchains," in *Proc. 18th ACM/IFIP/USENIX Middleware Conf.*, 2017, pp. 15–16.

MACIÀ MUT-PUIGSERVER was born in Mallorca, in 1966. He received the degree in computer science from the Autonomous University of Barcelona, in 1990, and the Ph.D. degree from the University of Balearic Islands (UIB), in 2006.

In 1996, he joined the Department of Mathematics and Computer Science, UIB, as an Associate Lecturer, where he has taught subjects in different grade courses and masters. As a Researcher, he has been involved in European-Funded Research Projects and some Spanish-Funded Research Projects. He is also a member of the Research Group Security and Electronic Commerce (SECOM), UIB, and of the ARES Consolider Excellence Network, a granted network in security and privacy with the objective of encouraging the cooperation between researchers. His current research interests include design of secure protocols, secure e-commerce, mobile applications, privacy-preserving applications, and applied cryptography. He has an active pace of publications on these topics in international journals, conferences, and book chapters. His recent research published is focused on the incorporation of blockchain technologies in secure and e-commerce applications.

MIQUEL A. CABOT-NADAL was born in Mallorca, in 1975. He received the Computer Engineering degree from the Open University of Catalonia (UOC), in 2009, and the master's degree in information and communications technologies (ICT) from the University of the Balearic Islands (UIB), in 2011, where he is currently pursuing the Ph.D. degree in information and communications technologies. His research interests include security, blockchain, and electronic commerce. He has a few conference and journal publications.

M. MAGDALENA PAYERAS-CAPELLÀ was born in Mallorca, in 1973. She received the degree in telecommunication engineering from the Polytechnic University of Catalonia (UPC), in 1998, and the Ph.D. degree in computer science from the University of the Balearic Islands (UIB), in 2005. Her Ph.D. dissertation was awarded with the COIT, Engineers Spanish Association, Best Ph.D. Thesis, in 2005.

She holds several positions with the Department of Mathematics and Computer Science, University of the Balearic Islands, since 1998. She is currently an Associate Professor and the Head of the Telecommunications Engineer Master. She has taught subjects in grade courses and in several masters (Information and Telecommunications Technologies Master, Information Technologies Master, Telecommunication Engineer Master, and Security in Information and Communications Master). She has directed a Ph.D. Thesis, in 2013, and is currently directing another one. She has participated in two European-Funded Research Projects, Five Spanish-Funded Research Projects, including a Consolider–Ingenio Project having been P.I. of one of them, 2014–2018, and three Balearic Islands-Funded Research Projects. She is a member of the Research Group SECOM and of the ARES Consolider Excellence Network, a granted network in security and privacy with the objective of encouraging the cooperation between researchers. She has participated in the creation of two spin-offs: the company KEYRON, dedicated to telemedicine, established in 2006, and the company GEDOCU IT Consulting dedicated to documental management, in 2016. Her research interests include security in communications networks, electronic payments, design of protocols for electronic commerce and secure applications, privacy-preserving applications, electronic ticketing, uses of cryptography, traffic management protocols, and technical-legal aspects of electronic commerce. Within these lines of research, she has had an active pace of publications in international journals and in international and national conferences. She is the author of more than ten articles published in international journals, included in Journal Citation Reports (JCR) in the last decade. She is the author of five book chapters and a coauthor of a patent (UIB-URV) about secure transferable electronic tickets. In the near future, her research intends to be focused on the integration of blockchain technologies in secure and privacy-preserving applications, especially those applications related to electronic commerce, touristic services, and smart mobility.

Dr. Payeras-Capellà has been a member of the Scientific and Organizing Committee in several international congresses and has wined three best paper awards for papers presented in conferences.