# Fair Signature Exchange via Delegation on Ubiquitous Networks

Q. Shi[1] (*Corresponding Author*), N. Zhang[2], and M. Merabti[1]

[1] School of Computing & Mathematical Sciences

Liverpool John Moores University

Byrom Street, Liverpool L3 3AF, UK

Email: {Q.Shi, M.Merabti}@ljmu.ac.uk

Tel: +44 (0) 151 231 2272

Fax: +44 (0) 151 207 4594

[2] School of Computer Science

The University of Manchester

Oxford Road, Manchester M13 9PL, UK

Email: ning.zhang@manchester.ac.uk

## Abstract

This paper addresses the issue of autonomous fair signature exchange in emerging ubiquitous (u-) commerce systems, which require that the exchange task be delegated to authorised devices for its autonomous and secure execution. Relevant existing work is either inefficient or ineffective in dealing with such delegated exchange. To rectify this situation, this paper aims to propose an effective, efficient and secure solution to the delegated exchange to support the important autonomy feature offered by u-commerce systems. The proposed work includes a novel approach to symmetric-key based verifiable proxy encryption to make the exchange delegation flexible, efficient and simple to implement on resource-limited devices commonly used in u-commerce systems. This approach is then applied to design a new exchange protocol. An analysis of the protocol is also provided to confirm its security and fairness. Moreover, a comparison with related work is presented to demonstrate its much better efficiency and simplicity.

**Key words:** Ubiquitous computing, Fair exchange, Signature, Communication protocol.

## 1. Introduction

The advance of ubiquitous computing technology enables everyday objects such as refrigerators and toasters to be augmented with information processing capabilities to offer ubiquitous network platforms on which to build smart integrated applications and services. This opens up great opportunities for pressing current electronic or mobile (*e*/*m*-) commerce technologies forward to provide seamless and intelligent business services from anywhere at anytime, which is also called ubiquitous (u-) commerce. While u-commerce greatly enhances the quality of life for individuals and families, its systems typically involve distributed and autonomous operations running on much open, dynamic and resource-diversified ubiquitous networks. These features are making the system security protection very challenging. Without proper security assurance, the wide acceptance and deployment of u-commerce would not become reality.

One of the important security challenges for u-commerce systems is about how to fulfil fair signature exchange. Such exchange means that two parties (e.g. individuals, companies or systems) can exchange their valuable digital signatures for agreed commercial transactions such as contract signing over networks without one party being disadvantaged by the other during the exchange process. More specifically, the fairness requires that either each party, or neither of them, can get the expected signature from the other party at the end of the exchange, which is also referred to as strong fairness [1]. This requirement is essential for preventing one party from deceptively

gaining business or financial advantages over the other.

To conduct a fair signature exchange, the existing work normally simplifies the exchange settings by assuming that decision making responsibilities for the exchange rest with the parties involved in the exchange and trusted computing devices are employed to execute the exchange. This simplification leads to a more focused issue of how to exchange the agreed signatures fairly without concerns about the environment in which the exchange is conducted. Accordingly the solutions developed under such assumptions are applicable to less intelligent e/m-commerce systems with fairly static exchange scenarios and settings.

However, emerging u-commerce systems are posing new challenges to fair signature exchange owing to their more complex features such as high autonomy, distributability and heterogeneity. To illustrate such features, we present the following example about a potential u-commerce setting for a user Alice and her smart home, which is derived from the example given in [20]:

- Alice is informed at work that her request for changing to a more rewarding job within her organisation has been granted. She is delighted with the news, and decides to invite her parents for dinner in her house in the evening to tell them the good news at the dinner table.
- She uses her smart phone to prepare a list of groceries needed for the dinner and sends it to her home manager - a software agent running on one of her home computing devices.
- The home manager responds to the request by initiating the following tasks:
  - Checks the RFID (radio frequency identification) enabled fridge and cupboards in the kitchen in reference to the received grocery list to decide what to buy, asks price quotations from nearby supermarkets, and sends a purchase order to the supermarket, offering the best deal, for home delivery in the afternoon;
  - Inspects the networked heating facilities fuelled by either solar power or gas controlled by a "pay as you go" meter, finds that the fuel supply is insufficient to keep the house warm due to cold weather, and thus decides to buy additional credits on-line from a gas supplier to top up the gas meter.
- Alice arrives at home after work, prepares the dinner with the delivered groceries, and shares the good news and nice meal with her parents in the warm house.

This application scenario involves the following signature exchanges:

(1) Alice's payment authorisation signature on the grocery order is exchanged for the supermarket's signature on a digital receipt for the paid groceries;

(2) Alice's payment authorisation signature on the gas credit purchase is exchanged for the gas supplier's signature on a digital ticket of the purchased credits.

Clearly, the above example demonstrates that Alice's home system is autonomous in deciding what, from whom, when and how purchases should be made with regard to given policies or requests. In this case, it is crucial for Alice to delegate the signature generation and exchange to her agent - home manager. Otherwise, she would have to make herself available for signing signatures when they are required, as she often does not know beforehand what to sign, e.g. the gas credit purchase mentioned above. This would seriously hinder the system's efficiency and effectiveness, especially when a signature is needed but Alice is unavailable to sign it.

Additionally, the system makes use of heterogeneous computing facilities ranging from small microprocessors embedded in devices such as a gas meter to a big home computer. The system operations are highly distributed among home appliances, mobile devices and the Internet for collaborative smart decision-making and autonomous execution. Clearly these capabilities are essential for the provision of smart services, which are much more beneficial to users than those offered by the existing e/m commerce applications. However, such benefit also brings complication into the u-commerce system. Particularly, its operations are much more open, dynamic, inter-operative and autonomous. This does not match the assumptions mentioned earlier for the development of existing signature exchange solutions, as exchange decision is no longer directly made by a user and computing devices used for the exchange could be vulnerable to security attacks. Consequently the existing solutions are either ineffective or inefficient to properly handle signature exchange in u-commerce settings, as will be discussed further later. Hence more research is needed to devise more suitable solutions.

The focus of this paper is on how to delegate the signature exchange task to an autonomous agent(s) and ensure the exchange fairness and security, which are essential for the exchange in u-commerce settings. So far, a large number of protocols have been developed for fair exchange [1-5, 7-8, 14, 18-20, 22-25]. Nevertheless, they are hardly intended for the emerging autonomous exchange scenarios of u-commerce. These protocols are mainly based on verifiable signature encryption to achieve the exchange fairness. They can be divided into two categories in terms of the types of encryption. The majority of the protocols fall in the first category that employs public-key based verifiable signature encryption (e.g. [2-5]). The other category comprises the protocols built on symmetric-key based verifiable signature encryption (e.g. [19-20, 25]).

The public-key based verifiable signature encryption allows its task to be delegated to a chosen agent(s), namely, it is applicable to u-commerce systems. The reason for this is that a public key is used to verifiably encrypt a

signature, so the public key can be directly given to the agent for performing the signature encryption. However, the symmetric-key based verifiable signature encryption is unsuitable for its direct delegation to a chosen agent, i.e., it is not directly applicable to u-commerce systems. Since a secret key is employed for a verifiable symmetric encryption of a signature in this case, the delegation of the encryption task to a chosen agent would require directly assigning the secret key to the agent. This is undesirable because such assignment could increase the risk of the key being compromised in vulnerable u-commerce operating environments.

Although the public-key based verifiable signature encryption is applicable to u-commerce systems, it is mathematically much more complex and computationally less efficient than the symmetric-key based verifiable signature encryption [25], which will be discussed further in Section 7. On the other hand, while the symmetric-key based verifiable encryption offers much better simplicity and efficiency, most of its solutions do not support the delegation capability, which makes them ineffective for u-commerce systems. A recent effort has been made to rectify the problem [20], but the approach proposed is complex and inefficient, which diminishes the simplicity and efficiency advantages of the symmetric-key based encryption.

The above weaknesses of the existing protocols have motivated us to propose a novel approach to symmetric-key based verifiable proxy encryption for the whole family of discrete logarithm based signature schemes [16] (e.g., DSA [12], ElGamal [6] and Schnorr [17]) in this paper. The approach is then utilised to formulate a new autonomous fair signature exchange protocol for u-commerce systems. The symmetric proxy encryption here means that given a long-term symmetric key $k_A$, a party $P_A$ generates a short-term symmetric encryption key $\kappa_A$ from $k_A$ and then delegates its task of verifiable symmetric encryption with $\kappa_A$ to a chosen agent $A_A$ running on a device connected to $P_A$'s ubiquitous network. Such delegation must ensure that the possession of $\kappa_A$ by $A_A$ does not permit $A_A$ to discover $k_A$. This assurance enables $A_A$ to execute the encryption on $P_A$'s behalf while averting the disclosure of $k_A$ in case $A_A$ is compromised.

Another important feature of the new approach is the verifiable symmetric encryption of a signature key to be used for the generation of a signature on $P_A$'s behalf by a designated party. Normally the construction of a key is mathematically simpler than that of a signature generated using the key. It should therefore be easier to build the verifiability of an encrypted signature key than that of an encrypted signature. This feature helps to simplify the new approach and improve its efficiency, which will become clear later in the paper.

However, the signature key could be subject to abuse if it is disclosed, as the key can be used illegitimately to yield any signature with $P_A$ bearing the responsibility. To prevent the key abuse, the new approach incorporates a

one-time property into the formation of the key so that the key is only valid for one specified document. In other words, the key is invalid for signing any other documents. By combining the one-time key with its verifiable symmetric encryption, the new approach is able to secure the signature exchange in a simple and efficient manner, as will be detailed in Section 4.

The main novel contribution of this paper comes from the verifiable symmetric proxy encryption of one-time signature keys, which is efficient, easy to implement, and flexible for application to a range of discrete logarithm based signature schemes. The new encryption preserves the good efficiency and simplicity attributes of existing verifiable symmetric encryption while adding the capability of encryption delegation. The one-time signature keys not only prevent their misuse for unauthorised purposes but also offer strong application flexibility. The existing work on verifiable encryption introduces different ways for different signature schemes, which complicates its implementation. The encryption of signature keys instead of signatures significantly weakens its dependency on signature schemes. This enables the new approach to be implemented for the family of discrete logarithm based signature schemes rather than just one of them. The above merits make the approach more suitable for operating on small resource-limited devices, which is essential for u-commerce systems.

In the rest of the paper, we will state the exchange settings for the proposed proxy encryption in Section 2. An introduction to the signatures to be used in this paper will be provided in Section 3. The new approach to the proxy encryption will be defined in Section 4. Based on this approach, Section 5 will propose the design of the protocol mentioned earlier for autonomous fair signature exchange in u-commerce. The protocol will be analysed to demonstrate its security strength in Section 6, and compared with related work to show its merits in Section 7. Finally, Section 8 will conclude the paper and point out future work.

## 2. Required Signature Exchange

In this section, we present the exchange scenario and assumptions on which our proposed approach is based, the fairness requirement which the approach needs to meet for the given exchange, and a summary of notations to be used for the presentation of the approach.

### 2.1. Exchange Settings and Assumptions

The case of fair signature exchange covered in this paper is described below:

- *Signature exchange agreement*: Let a party $P_A$ be a user (or owner) of a private ubiquitous computing network and run a u-commerce system on the network. This means that the system has its software components

distributed on various devices connected to the network. For simplicity, these components will be termed *agents* hereafter. Ideally the system should provide a decision-making capability for approving exchange cases with regard to given policies and/or requests from $P_A$, and then offer effective solutions for executing the approved exchanges autonomously. The focus of this paper is only on the design of such a solution, since the development of the decision-making capability itself is an area requiring further research.

To devise the exchange solution, we assume that $P_A$ has appointed an agent $A_A$, called a *proxy exchanger*, in charge of performing all the tasks of an agreed exchange except for the generation of $P_A$'s signatures. $P_A$ has also designated the authority of its signature generation to a group of different agents, named *proxy signers*, which are distributed on various devices and only permitted to jointly produce signatures on $P_A$'s behalf. This means that $A_A$ has to request these signers for the creation of necessary signatures for an exchange.

The reason for separating the signature generation from the other tasks of the exchange is to avoid a single agent being able to perform the entire exchange. Since ubiquitous networks are susceptible to security attacks, the use of such an agent would make the proposed solution vulnerable in case the agent is compromised. For the same reason, the responsibility of signature generation is shared among the chosen proxy signers to prevent the private signature key from being misused by an adversary to produce $P_A$'s signatures for fraudulent purposes with $P_A$ facing the consequences. A threshold signature scheme (e.g. [21]) can be adopted for the joint signature generation. Such a scheme can strengthen both security and reliability, because it is much harder to compromise multiple agents or devices and the scheme still works even when some agents fail to collaborate.

However, for simplicity, we only employ one agent $A_A$ to play the proxy exchanger role in this paper. As will be made clear in Section 4, $A_A$ only holds a short-term key for signature encryption. The key is not as important as the private signature key. The work presented in this paper can be extended to allow multiple agents to jointly serve the exchanger role for better security and reliability. Such extension will be considered in our future work.

As the focus of this paper is on the proposed verifiable encryption, the details of the aforementioned joint signature generation will not be presented in this paper. We simply assume that $A_A$ can receive $P_A$'s signatures from the signers for approved signature exchanges.

For a signature exchange, we suppose that $A_A$ is informed by $P_A$'s decision-making agent(s) that an agreement

has been reached with another party $P_B$ to exchange a signature of $P_A$ on a document $D_A$ for a signature of $P_B$ on a document $D_B$. Here, the design of the decision-making agent is beyond the scope of this paper. Also the agent passes the information on $D_A$ to the signers, so they know what document they should sign when $A_A$ makes a request for the signature.

Using the gas credit purchase discussed in Section 1 as an example, the above exchange means that an appointed agent (i.e. $A_A$) of Alice (i.e. $P_A$) obtains a payment authorisation signature on the gas credit purchase request (i.e., $D_A$) from the signers, and then exchanges it for a signature of the gas supplier (i.e. $P_B$) on a digital ticket (i.e., $D_B$).

Additionally, $P_B$ normally represents an online merchant or service provider with more centralised powerful computing facilities to run its business, although $P_B$ could delegate its tasks to chosen agents. For easy presentation, this paper assumes that $P_B$ directly carries out the agreed signature exchange. The work to be presented later is equally applicable to the case where $P_B$ uses the delegation, which will become clear later.

- *Agreed trusted third party* (*TTP*): Suppose that $A_A$ and $P_B$ do not trust each other to perform the agreed exchange honestly. In this case, we assume that they have agreed to employ an off-line TTP $P_T$ for helping them to complete the exchange fairly when required. The off-line $P_T$ means that $P_T$ is not needed when the exchange is completed correctly, and it is activated otherwise to recover necessary information for the fair exchange completion.

- *Public key certification*: Let each party $P_I \in \{P_A, P_B, P_T\}$ involved in the exchange possess a pair of public and private keys, $(u_I, r_I)$, to be defined in Section 3.2. To ensure the authenticity of signatures signed with private key $r_I$, we assume that public key $u_I$ has been certified by a certification authority (or CA) and known by all the other parties.

- *Security threats*: Suppose that there is an adversary keen on breaking the security of the agreed signature exchange for various purposes such as financial fraud. The adversary has sufficient resources to intercept $A_A$'s communications with the signers and $P_B$ and even compromise some of these agents. If an agent is compromised, we assume that all the information possessed by the agent is exposed to the adversary. The main security threat considered in this paper is that the adversary attempts to utilise the intercepted information and compromised agents to obtain $P_A$'s private/secret keys.

  In the event where some agents including $A_A$ are compromised, we assume (a) there is no collusion between

compromised $A_A$ and any other party involved in an exchange with $A_A$ (or $P_A$), and (b) in any case, the adversary cannot gain $P_A$'s private key from any compromised signers and does not have enough compromised signers to jointly generate $P_A$'s signatures by themselves.

Assumption (a) above is essential because no fair exchange approach would work if the host used by one party for an exchange colludes with the other party involved in the same exchange. More specifically, since the compromised host handles necessary signatures for the exchange, the other party could directly manipulate the host to obtain wanted signatures, regardless of the security of the exchange approach employed. The assumption can be relaxed by utilising multiple agents to jointly play the role of $A_A$ as mentioned earlier. Assumption (b) is achievable by employing sufficient signers for private key distribution and joint signature generation.

### 2.2. Exchange Fairness Requirement

The signature exchange case described in the previous sub-section needs to satisfy the following requirement:

*At the end of the exchange, if agent $A_A$* (or party $P_A$) *has obtained a signature of party $P_B$ on agreed document $D_B$ or can obtain the signature through TTP $P_T$, then $P_B$ has obtained a signature of $P_A$ on agreed document $D_A$ from $A_A$ or can obtain the signature through $P_T$, and vice versa.*

In other words, the above requirement ensures that either each of $A_A$ and $P_B$, or neither of them, can obtain the other's valid signature on the agreed document. This is equivalent to the strong fairness stated in [1]. Such fairness is preferable because any dispute about the exchange can be resolved among $A_A$ (or $P_A$), $P_B$ and $P_T$ themselves. This can avoid the use of external legal means such as a court of justice for the dispute resolution, which is not only costly but also infeasible in many cases, particularly when $P_A$ and $P_B$ are in different countries.

### 2.3. Notation List

To facilitate the understanding of the proposed approach, Table 1 provides a list of main variables and functions to be used for its specification.

## 3. Signatures

As pointed out in Section 1, the work proposed in this paper is targeted at the family of discrete logarithm based signature schemes [16], rather than a particular signature scheme, for better applicability. To present the work, it is necessary in this section to provide an introduction to the concept of proxy signatures [10] and also an overview

of the Schnorr signature scheme [17]. The proxy signatures will be used by party $P_A$'s designated agents to sign agreed documents on $P_A$'s behalf. The Schnorr scheme will be adopted to construct proxy keys for the generation of discrete logarithm based proxy signatures.

| Item | Description |
|---|---|
| $P_I$, $u_I$, $r_I$ | Party $P_I \in \{P_A, P_B, P_T\}$, and its public & private keys (see Sections 2.1 & 3.2) |
| $q$, $p$, $g$ | Public parameters for discrete logarithm based signatures (see 3.2) |
| $H(x)$ | Generate a hash of data item $x$ (see 3.2) |
| $D_A$, $D_B$, $d_A$, $d_B$ | Documents of $P_A$ and $P_B$, and their hashes $d_A = H(D_A)$ and $d_B = H(D_B)$ (see 2.1) |
| $A_A$ | An agent of party $P_A$, appointed as a proxy exchanger (see 2.1) |
| $S = Sign(d, r)$ | Generate a signature $S$ on hash $d$ with private key $r$ (see 3.1) |
| $v = Verify(S, d, u)$ | Verify $S$ on $d$ using public key $u$, with outcome $v = yes$ / $no$ (see 3.1) |
| $x \| y$ | Denote the concatenation of data items $x$ and $y$ (see 3.2) |
| $k_A$, $\delta_A = g^{k_A} \bmod p$ | $P_A$'s long-term key shared with $P_T$ and its public parameter (see 4.2) |
| $C_A = (\iota_A, \delta_A, s_T)$ | $P_A$'s certificate issued by $P_T$ for key $k_A$ (see 4.2) |
| $x_A$, $y_A = g^{x_A} \bmod p$ | $P_A$'s random number ($< q$) and its public parameter (see 4.2) |
| $\kappa_A = (k_A + x_A) \bmod q$ | $P_A$'s proxy encryption key assigned to $A_A$ (see 4.2) |
| $w_A$, $\varkappa_A$, $h_A = H(\varkappa_A)$ | $P_A$'s warrant, session header and its hash (see 4.2 & 5) |
| $\gamma_A$, $\eta_A$ | $P_A$'s Schnorr signature with random $\alpha_A < q$: $\beta_A = g^{\alpha_A} \bmod p$, $\eta_A = H(h_A \| d_A \| \beta_A \| y_A \| w_A)$, and $\gamma_A = (\eta_A \times r_A + \alpha_A) \bmod p$, with $\gamma_A$ as a private proxy signature key for $A_A$ (see 4.3) |
| $\tau_A = (\gamma_A + x_A) \bmod q$ | $P_A$'s private proxy signature key for $P_T$ (see 4.3) |
| $e_A = (\tau_A + k_A) \bmod q$ | Encryption of $\tau_A$ with $k_A$ (see 4.3) |
| $K_A = (e_A, \eta_A, y_A, w_A)$ | Verifiable encryption of $\tau_A$ (see 4.3) |
| $o=Check(h_A,d_A,K_A,C_A)$ | Check $K_A$ and $C_A$ using $h_A$ and $d_A$ with outcome $o = yes$ / $no$ (see 4.4) |
| $S_A$, $\varsigma_A$ | $P_A$'s proxy signatures yielded by $A_A$ and $P_T$ respectively (see 5) |
| $S_B$ | $P_B$'s signature (see 5) |

Table 1. List of main variables and functions used in this paper.

### 3.1. Proxy Signatures

To achieve the exchange autonomy discussed in Section 1, it is essential for $P_A$ to delegate its power for signature generation and exchange to chosen agents. To enforce the signing delegation, proxy signature techniques can be applied to enable the designated agents to generate signatures on $P_A$'s behalf.

Proxy signatures are classified into full delegation, partial delegation and delegation by warrant [10]. The full delegation allows a proxy signer to share a private key of an original signer, namely, a proxy or original signature can be produced by either of the two signers. In partial delegation, an original signer yields a private proxy key from its original private key and assigns the proxy key to a proxy signer so that signatures generated by the proxy and original signers are different. The delegation by warrant lets an original signer issue a warrant to a proxy signer for the generation of proxy signatures on its behalf with a designated key different from the original signer's private key.

Proxy signatures can also be further divided into two types: proxy-unprotected and proxy-protected [10]. A proxy-unprotected signature means that it can be produced by either of the original and proxy signers. A proxy-protected signature implies that it can be yielded only by the proxy signer. To achieve this, the proxy signer normally incorporates its own private key in the generation of the proxy signature so that the original signer is unable to produce the same signature.

In this paper, we will only consider proxy-unprotected signatures with partial delegation due to the following reasons. Firstly, the full delegation allows an original private key of a party to be shared with its chosen agent (or device), which increases the risk of the key being compromised in vulnerable u-commerce environments. Secondly, the delegation by warrant is less efficient than the partial delegation [10]. Finally, it is unnecessary to adopt proxy-protected signatures. As will be shown in Section 4, a proxy signer is either exchanger $A_A$ chosen by party $P_A$, or TTP $P_T$ introduced in Section 2.1. Since $A_A$ runs within $P_A$'s own system, the responsibility for its protection rests with $P_A$. Although $P_T$ can be protected via its own signatures, it is undesirable to do this in order to avoid the validity of $P_A$'s proxy signatures replying on $P_T$'s signatures as argued in [3]. However, the approach proposed in Section 4 offers a different way for $P_A$ to distinguish between proxy signatures generated by $A_A$ and those produced by $P_T$, which is useful in case $P_A$ needs to identify the signer of an improperly yielded proxy signature.

For simplicity, proxy signatures will mean proxy-unprotected signatures with partial delegation hereafter. One way to produce such signatures is to let $P_A$ determine a private proxy signature key $\gamma_A$ based on its original private signature key $r_A$, and assign $\gamma_A$ to its chosen exchanger $A_A$ as a proxy signer. Here, the method must ensure that it

is hard for any other party to derive $r_A$ from $\gamma_A$. This means that even if $\gamma_A$ is compromised, $r_A$ is still secure for $P_A$ to use. $A_A$ will apply $\gamma_A$ to generate a proxy signature on a given document on $P_A$'s behalf. To minimise any abuse or misuse of $\gamma_A$, its validity is normally restricted to a particular purpose specified in an associated warrant. Similarly, another private proxy signature key $\tau_A$ can be created for $P_T$.

As mentioned earlier, the work presented in this paper is focused on the family of discrete logarithm signature schemes rather than a particular one. Thus, in the rest of the paper, we will not mention any specific scheme for proxy signatures and only use the following two functions to signify an agreed scheme for the generation and verification of a signature, respectively:

$$S = Sign(d, r), \text{ and } v = Verify(S, d, u).$$

Here, $Sign(\cdot)$ yields a signature $S$ signed on a document hash $d$ with a private key $r$. $Verify(\cdot)$ checks the correctness of signature $S$ on hash $d$ using a public key $u$. It assigns $v = yes$ if $S$ is correct, and $v = no$ otherwise.

Although no specific signature scheme will be mentioned for proxy signatures, the next sub-section will provide an overview of the Schnorr signature scheme [17] as it will be utilised for the creation of proxy signature keys in Section 4.

### 3.2. Schnorr Signatures

Let $p$, $q$ and $g$ be three public parameters. $p$ is a large prime, $q$ is a large prime factor of $p$ - 1, and $g$ is a number between 1 and $p$ such that $g^q \bmod p = 1$. To generate a pair of private and public keys, a party $P_A$ chooses a random number $r_A < q$ as its private key, and calculates $u_A = g^{r_A} \bmod p$ as the corresponding public key.

To produce signatures, a secure one-way hash function such as SHA-2 [13] is needed, which is denoted as $H(x)$. It should possess the following properties: (a) for any $x$, it is easy to compute $H(x)$; (b) given $x$, it is hard to find $x'$ ($\neq x$) such that $H(x) = H(x')$; and (c) given $H(x)$, it is hard to compute $x$.

To yield a signature on an agreed document $D_A$, $P_A$ picks a random number $a_A < q$ to compute:

$$b_A = g^{a_A} \bmod p,\ m_A = H(H(D_A) \parallel b_A), \text{ and } s_A = (m_A \times r_A + a_A) \bmod q.$$

Here, "$x \parallel y$" signifies the concatenation of data items $x$ and $y$.

The signature on $D_A$ is defined as $(s_A, m_A)$. Additionally, other items such as a time stamp could be added into $H(\cdot)$ for the signature generation if necessary.

Given document $D_A$ and signature ($s_A$, $m_A$) together with $P_A$'s public key $u_A$, a signature verifier can calculate:

$$b'_A = (g^{s_A} \times u_A^{-m_A}) \bmod p.$$

If $m_A = H(H(D) \| b'_A)$, then the signature is valid for $D$.

The Schnorr signature scheme can also be used to generate threshold signatures [21]. As stated in Section 2.1, the focus of this paper is on verifiable proxy encryption, so we simply assume that a secure threshold scheme is available for the joint generation of Schnorr signatures, without further discussions on its details.

## 4. Proposed Verifiable Symmetric Proxy Encryption

This section will define the details of the proposed new approach to verifiable proxy encryption based on the Schnorr signature scheme introduced earlier in Section 3.2.

### 4.1. An Overview of the New Approach

The proposed approach consists of the following stages:

1. *Short-term proxy encryption key creation*: For symmetric encryption, it is essential for two parties to share a secret key so that one party can use the key to encrypt sensitive data and the other party can utilise the same key to decrypt the encryption for the recovery of the original data. Thus we require that party $P_A$ and TTP $P_T$ have such a shared secret key $k_A$, which is for long-term use with a defined expiration date. $k_A$ will be used by $P_A$ to encrypt signatures or keys and by $P_T$ to decrypt them, and the need for this will become clear in the subsequent sub-sections.

   As explained in Section 2.1, $P_A$'s encryption task should be delegated to its exchanger agent $A_A$ due to the nature of autonomous u-commerce operations. This implies that $A_A$ has to get hold of $k_A$ or its altered form. Since $A_A$ may operate in a vulnerable ubiquitous network environment, there is a risk of security compromise to $A_A$, so it is undesirable to allow $A_A$ to directly possess $k_A$. Alternatively, an altered form of $k_A$ can be created for $A_A$, which is for specified short-term use and with restricted validity for security reasons. This is the option adopted by our proposed approach, which will be described in detail in Section 4.2. The altered key will be called a proxy encryption key and signified as $\kappa_A$. Obviously, the formation of $\kappa_A$ must ensure that any encryption with $\kappa_A$ can still be decrypted by $P_T$ with $k_A$, and $\kappa_A$ cannot be used to derive $k_A$, so that $k_A$ is still secure even when $A_A$ is compromised to cause the disclosure of $\kappa_A$. The introduction of $\kappa_A$ enables $A_A$ to perform the encryption on $P_A$'s behalf while protecting $P_A$'s long-term key $k_A$.

2. *One-time proxy signature key generation*: As stated in Section 2.1, $P_A$ needs to delegate its signing power to a group of signers for the joint generation of its signatures. However, the approach proposed in this paper does not directly use such a signature for an exchange. Instead, the signature is utilised to construct two different private proxy signature keys, denoted as $\gamma_A$ and $\tau_A$, for $A_A$ and $P_T$ respectively, which will be fully specified in Section 4.3. Note that these two proxy keys cannot be processed to derive $P_A$'s original private key $r_A$. $\gamma_A$ allows $A_A$ to yield a proxy signature of $P_A$ based on an agreed signature scheme for a direct exchange of $P_B$'s signature in normal situations. $\tau_A$ is used by $P_T$ to produce another proxy signature of $P_A$ for $P_B$ only in an abnormal case where $A_A$ and $P_B$ are unable to complete their exchange properly. For accountability, each of $A_A$ and $P_T$ should not know the other's private proxy key so that $P_A$ can identify which proxy signature is produced by whom. This helps $P_A$ to investigate the conduct of $A_A$ or $P_T$ in case a dispute about a proxy signature occurs.

   To prevent the proxy keys from being abused for illegitimate purposes, the keys need to possess the one-time property discussed in Section 1. In other words, the creation of the proxy keys must ensure that they are valid only for a specified exchange. Thus what $A_A$ and $P_T$ can do with the keys is to simply generate the expected proxy signatures. This reduces the reliance of our approach on the security of $A_A$ and $P_T$.

3. *Verifiable symmetric proxy encryption*: For the agreed exchange, private proxy signature key $\tau_A$ discussed above needs to be delivered securely to $P_T$ for its generation of $P_A$'s proxy signature for $P_B$ when needed. However, $P_B$ must be assured that $P_T$ indeed has the possession of $\tau_A$, before $P_B$ can release its signature to $A_A$ (or $P_A$). Otherwise, having transferred its signature to $A_A$, $P_B$ would not be able to obtain $P_A$'s proxy signature from $P_T$ if $A_A$ refuses to send $P_B$ its proxy signature yielded with proxy signature key $\gamma_A$. An effective way to provide the assurance to $P_B$ is to pass $\tau_A$ to $P_B$ in such a way that $\tau_A$ is encrypted to stop $P_B$ getting $\tau_A$, but $P_B$ can verify the encryption to confirm that correct $\tau_A$ is indeed in the encryption and $P_T$ is in the possession of the right decryption key for the recovery of $\tau_A$. Such encryption is named verifiable encryption. Only when $P_B$ requests $P_T$ to recover a proxy signature of $P_A$, $P_B$ passes encrypted $\tau_A$ to $P_T$. Clearly $P_T$ has no need to know $\tau_A$ in a normal exchange with no signature recovery.

   Note that $P_B$ is unable to abuse the above signature recovery process to gain advantages over $A_A$ (or $P_A$), as will be discussed in Sections 6.

   $A_A$ is assigned the responsibility of performing the above verifiable encryption using its proxy encryption key

$\kappa_A$ introduced earlier. The main challenges here are that $A_A$ is not allowed to know $\tau_A$ as discussed earlier but it needs to encrypt $\tau_A$, i.e. $A_A$ has to encrypt $\tau_A$ without seeing it, and also that the encryption has to be verifiable by $P_B$. To address the first challenge, our approach splits $\tau_A$ into two parts, one generated jointly by the signers and the other hidden as a factor of $\kappa_A$. This means that $A_A$ cannot assemble the two parts to form $\tau_A$ but can encrypt it by combining the first part with $\kappa_A$, as will be detailed in Section 4.3. To tackle the second challenge, our approach constructs the encryption of $\tau_A$ in a style similar to a Schnorr signature, but the encryption itself is not a valid signature of $P_A$. This helps to simplify the accomplishment of the verifiability of encrypted $\tau_A$ without disclosing $\tau_A$ and $\kappa_A$ to $P_B$.

In addition to the simplicity of the above encryption, the use of key $\tau_A$ instead of a proxy signature for the encryption makes our approach applicable to the whole family of discrete logarithm based signature schemes. This is because public and private keys used by these schemes share the key style defined in Section 3.2. Hence, the verifiable key encryption allows our approach to be independent of the individual signature schemes, and given such a scheme agreed by $A_A$ and $P_B$, it can be directly utilised to generate a proxy signature with $\tau_A$. Otherwise, these different schemes would require different methods for verifiable signature encryption, which could be costly to implement, particularly on resource-limited computing devices employed by u-commerce systems.

To facilitate a better understanding of the aforementioned keys, Fig. 1 shows these keys, their generators and recipients. For example, $P_A$ creates its private signature key $r_A$ and assigns its shares to the chosen signers for joint proxy signature production using a threshold signature scheme adopted by $P_A$.

In the rest of this section, we will elaborate the approach described above, including encryption key generation, verifiable proxy signature key encryption, encrypted key verification and signature recovery.

## 4.2. Encryption Key Generation

For long-term key $k_A$ shared between $P_A$ and $P_T$ discussed in the previous sub-section, $k_A$ needs to be certified by $P_T$. The certification serves two purposes. First, it will be used by $P_B$ to verify the encrypted proxy signature key $\tau_A$ of $P_A$ to confirm that $P_T$ has the right key to decrypt it, which will be detailed in the next sub-section. Secondly, the certificate allows $P_T$ to re-calculate $k_A$ when needed, so there is no need for $P_T$ to store $k_A$ so as to reduce resource demand on $P_T$.

To issue the certificate, $P_T$ defines a certificate header or label $\iota_A$ including $P_A$'s identity, public key and a valid period for the certificate. $P_T$ then uses its private key $r_T$ to compute:

$$k_A = H(r_T \,||\, \iota_A) \bmod q, \text{ and } \delta_A = g^{k_A} \bmod p.$$

Here, public parameters $p$, $q$ and $g$ were defined in Section 3.2.

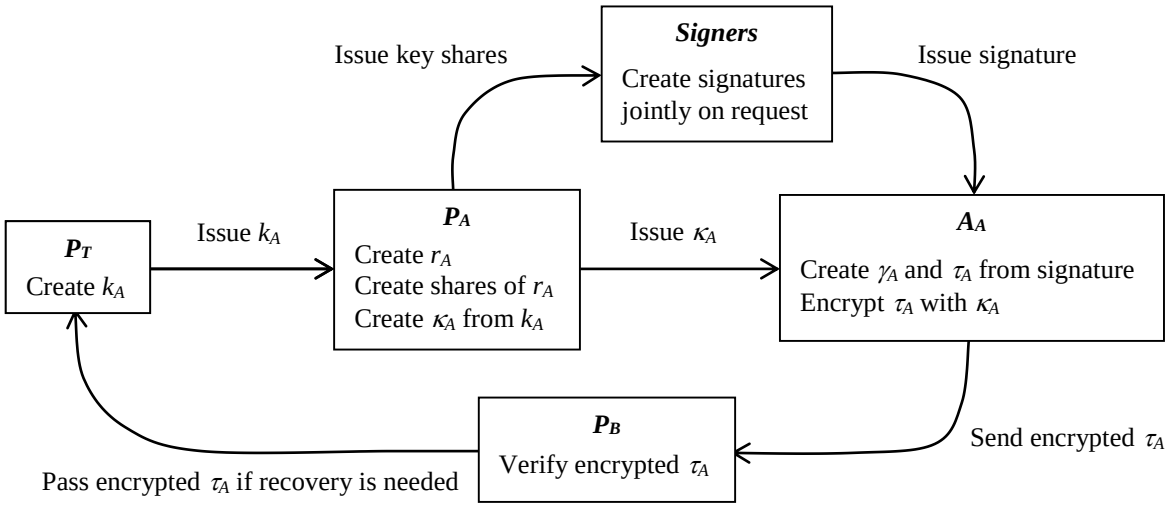


Fig. 1. Relationships among keys, their generators and recipients for the proposed approach.

$k_A$ and $\delta_A$ appear like a pair of private and public Schnorr keys introduced in Section 3.2. However, they are not valid for the generation of $P_A$'s signatures and used only for verifiable proxy signature key encryption to be presented in the next sub-section. From the formation of $k_A$, it is clear that given $\iota_A$, $P_T$ can easily re-produce $k_A$, so $P_T$ does not keep any information about $k_A$. Also $k_A$ is reusable for different exchanges within the specified valid period. $\delta_A$ will be applied by $P_B$ to check that the verifiable encryption involves $k_A$ in the form which $P_T$ can decrypt to recover the signature key. Obviously the integrity of $\delta_A$ must be guaranteed. Otherwise, the use of a wrong $\delta'_A$ together with its associated $k'_A$ would mean that $P_T$ could not get a right key for the decryption.

To ensure the integrity of $\delta_A$, $P_T$ certifies it as follows:

$$C_A = (\iota_A, \delta_A, s_T), \text{ with its signature } s_T = Sign(H(\iota_A \,||\, \delta_A), r_T).$$

The above function $Sign(\cdot)$ was introduced in Section 3.1. It represents $P_T$'s signature on items $\iota_A$ and $\delta_A$. This indicates that any alteration to certificate $C_A$ would lead to a verification failure of signature $s_T$, namely, the certificate including $\delta_A$ is invalid.

Finally, $P_T$ transfers key $k_A$ and certificate $C_A$ to $P_A$ securely by encryption, and deletes both of them afterwards.

Having received $k_A$ and $C_A$ from $P_T$, $P_A$ can confirm their correctness by checking that $\iota_A$ and $\delta_A$ in $C_A$ are correctly signed by $s_T$, $\delta_A$ is equal to $g^{k_A} \bmod p$, and $\iota_A$ contains right information.

Based on $k_A$ received, $P_A$ needs to generate a proxy encryption key $\kappa_A$ discussed earlier in Section 4.1 for the delegation of its verifiable encryption task to its exchanger agent $A_A$ without disclosing $k_A$ to $A_A$. To prevent the delegation from being misused for illegitimate purposes, $P_A$ has to clearly define a warrant $w_A$ that sets out terms and conditions for the delegation including the proxy signature generation outlined in the previous sub-section. $w_A$ includes information such as $P_A$'s identity and public key, a valid period and permitted uses of the delegation, and acceptable forms of proxy signatures to be discussed further in Section 4.5. If the delegation goes beyond the scope stated in $w_A$, $P_A$ will not take any responsibility for the misbehaviour. Evidently $w_A$ must be embedded in each proxy signature of $P_A$ to confine its validity to the purposes specified in $w_A$.

After the definition of $w_A$, $P_A$ creates proxy encryption key $\kappa_A$ by picking a random number $x_A < q$ to calculate:

$$y_A = g^{x_A} \bmod p, \text{ and } \kappa_A = (k_A + x_A) \bmod q.$$

$P_A$ then assigns ($\kappa_A$, $y_A$, $w_A$) together with certificate $C_A$ to $A_A$ securely. This allows $A_A$ to perform verifiable proxy encryptions with $\kappa_A$ for different exchanges, as long as $w_A$ is not expired. Since $A_A$ is not given $x_A$ and cannot get $x_A$ from $y_A$, it is hard for $A_A$ to obtain $k_A$ from given $\kappa_A$. This will be further justified by a detailed security analysis in Section 6.

As assumed in Section 2.1, $P_A$ has a set of signers appointed for joint signature generation on $P_A$'s behalf. $P_A$ also needs to pass ($y_A$, $w_A$) to each of these signers so that $y_A$ and $w_A$ can embedded in the generation of each signature. In other words, the signature is valid only for the purposes stated in $w_A$. Moreover, the inclusion of $y_A$ in the signature is intended to deter $A_A$ from maliciously altering $\kappa_A$. Since both $k_A$ and $x_A$ in $\kappa_A$ are now fixed via $C_A$ and the signature respectively, the alteration can be easily spotted, which will become clear in Section 4.4. Thus $\kappa_A$, $y_A$ and $w_A$ are bond together.

Note that $\kappa_A$ could take the form of a Schnorr signature, e.g. $\kappa_A = (H(w_A \parallel y_A) \times k_A + x_A) \bmod q$, to bind $\kappa_A$ to warrant $w_A$ and number $y_A$. However, it would add more computation, i.e. $\delta_A^{-H(w_A \parallel y_A)}$, to the verification to be presented in Section 4.4, because the verification would have to check that $y_A = (g^{\kappa_A} \times \delta_A^{-H(w_A \parallel y_A)}) \bmod p$. Hence the way discussed earlier for getting $\kappa_A$, $y_A$ and $w_A$ bond together is more efficient.

### 4.3. Verifiable Symmetric Proxy Encryption of Proxy Signature Keys

We now propose methods for creating private proxy signature keys $\gamma_A$ and $\tau_A$ discussed in Section 4.1 for the delegation of $P_A$'s signing power to $A_A$ and $P_T$ for proxy signature generations respectively, and then getting $\tau_A$ delivered to $P_T$ via $P_B$ by verifiable encryption. These methods are crucial for autonomous fair signature exchange. The signing delegation helps to achieve the autonomy, and the verifiable encryption to fulfil the fairness.

To yield $\gamma_A$ and $\tau_A$ for an agreed signature exchange, $A_A$ first gets hold of the hash $d_A$ of its agreed document $D_A$ (i.e. $d_A = H(D_A)$) and another hash $h_A$ of other information related to the current session of the exchange, which will be defined in Section 5. $A_A$ then passes $h_A$ and $d_A$ to the designated signers to request them for the generation of a Schnorr signature on $h_A$ and $d_A$. Here we assume that upon receipt of $A_A$'s request, the signers are able to assess its validity with respect to possessed warrant $w_A$, and produce the signature for $A_A$ only if the request is valid. In the case of the valid request, the signers jointly generate a Schnorr signature ($\gamma_A$, $\eta_A$) for $A_A$ on $P_A$'s behalf, which is expressed as:

$$\beta_A = g^{\alpha_A} \bmod p,\ \eta_A = H(h_A \| d_A \| \beta_A \| y_A \| w_A), \text{ and } \gamma_A = (\eta_A \times r_A + \alpha_A) \bmod q.$$

Here, $\alpha_A$ is a random number $< q$, and $y_A$ together with $w_A$ was assigned to each signer in Section 4.2.

Having received signature ($\gamma_A$, $\eta_A$) from the signers securely, $A_A$ keeps $\gamma_A$ as its own private proxy signature key for later proxy signature generation. The public proxy signature key $\mu_A$ associated with $\gamma_A$ can be computed below with $P_A$'s public key $u_A$:

$$\mu_A = (u_A^{\eta_A} \times \beta_A) \bmod p.$$

This is because:

$$\mu_A = g^{\gamma_A} \bmod p = g^{\eta_A \times r_A + \alpha_A} \bmod p = ((g^{r_A})^{\eta_A} \times g^{\alpha_A}) \bmod p = (u_A^{\eta_A} \times \beta_A) \bmod p.$$

Additionally, $A_A$ applies $\gamma_A$ to construct another private proxy signature key $\tau_A$ for TTP $P_T$, which will be used by $P_T$ to yield a proxy signature on $P_A$'s behalf when $P_T$ is requested for signature recovery in an abnormal case of exchange, as will be detailed in Section 4.5. $\tau_A$ is defined below with $x_A$ and $y_A = g^{x_A} \bmod p$ introduced in Section 4.2 for the creation of proxy encryption key $\kappa_A$:

$$\tau_A = (\gamma_A + x_A) \bmod q.$$

Its corresponding public proxy signature key can be calculated as:

$$\upsilon_A = (u_A^{\eta_A} \times \beta_A \times y_A) \bmod p.$$

This is due to the following relationship:

$$\upsilon_A = g^{\tau_A} \bmod p = g^{\gamma_A + x_A} \bmod p = g^{\eta_A \times r_A + \alpha_A + x_A} \bmod p = (u_A{}^{\eta_A} \times \beta_A \times y_A) \bmod p.$$

Obviously, $A_A$ is unable to directly produce $\tau_A$ as $A_A$ does not know $x_A$. Instead, $A_A$ generates an encrypted form of $\tau_A$ with key $\kappa_A$:

$$e_A = (\gamma_A + \kappa_A) \bmod q.$$

Since $e_A$ can be expressed as $e_A = (\gamma_A + \kappa_A) \bmod q = (\gamma_A + x_A + k_A) \bmod q = (\tau_A + k_A) \bmod q$ with shared key $k_A$ defined via $C_A$ in Section 4.2, $e_A$ is in effect the encryption of $\tau_A$ with $k_A$. As will be shown in the subsequent sub-sections, $e_A$ is verifiable by any party to confirm that it contains correct $\tau_A$, and $\tau_A$ is recoverable from $e_A$ only by $P_T$ because given certificate $C_A$, only $P_T$ can retrieve $k_A$ from $C_A$ and then decrypt $e_A$ with $k_A$ for $\tau_A$. In other words, $e_A$ along with its related items represents a verifiable symmetric proxy encryption of proxy signature key $\tau_A$ with encryption key $k_A$, and $e_A$ can be decrypted by $P_T$ using the same key $k_A$. Note that $e_A$ itself cannot be used as a valid proxy signature key of $P_A$, and it is hard for any other party to derive $\gamma_A$ from $e_A$ without knowing $k_A$, which will be justified in Section 6.

Recall the discussion in Section 4.1 that the two private proxy signature keys $\gamma_A$ and $\tau_A$ defined above should be one-time keys. This will be explained in detail in Section 4.5.

After the generation of $e_A$, $A_A$ defines:

$$K_A = (e_A, \eta_A, y_A, w_A)$$

as its verifiably encrypted proxy signature key $\tau_A$. Here, $A_A$ received $(y_A, w_A)$ along with encryption key $\kappa_A$ and certificate $C_A$ from $P_A$ in Section 4.2, and $\eta_A$ together with signature key $\gamma_A$ from the signers earlier.

$A_A$ then transfers $(K_A, C_A)$ along with the other related information to $P_B$ in order to exchange for an expected signature $S_B$ of $P_B$. If $A_A$ receives correct $S_B$ from $P_B$ as an acknowledgement to its acceptance of $(K_A, C_A)$, $A_A$ applies private proxy signature key $\gamma_A$ to yield a required proxy signature $S_A$ on $h_A$ and $d_A$ for $P_B$ using the agreed signature scheme $Sign(\cdot)$ stated in Section 3.1, i.e. $S_A = Sign(H(h_A \parallel d_A), \gamma_A)$. Note that $A_A$ produces $S_A$ by itself without any involvement of the signers. $A_A$ then sends $S_A$ to $P_B$. The reception of valid $S_A$ by $P_B$ indicates the successful completion of the exchange with no need for any involvement of $P_T$. This process will be discussed further in Section 5.

### 4.4. Encrypted Signature Key Verification

Having received $K_A = (e_A, \eta_A, y_A, w_A)$ and $C_A = (\iota_A, \delta_A, s_T)$ together with agreed document hash $d_A$ and related information hash $h_A$ from $A_A$, $P_B$ must examine the correctness of $K_A$ and $C_A$. The examination of $C_A$ is intended to ensure that given valid $C_A$, $P_T$ can recover key $k_A$ associated with $\delta_A$ as described in Section 4.2. The verification of $K_A$ serves to confirm that $e_A$ is indeed the symmetric encryption of correct signature key $\tau_A$ with the encryption key (i.e. $k_A$) linked to $\delta_A$. Collectively, correct $K_A$ and $C_A$ assure $P_B$ that $P_T$ can recover $k_A$ from $C_A$ and then apply it to decrypt $e_A$ for $\tau_A$, in case a signature recovery is required. The possession of $\tau_A$ enables $P_T$ to create a proxy signature of $P_A$ for $P_B$. This proves $A_A$'s commitment to the exchange. On the other hand, if either $K_A$ or $C_A$ is invalid, $P_B$ simply rejects them.

To check the validity of $C_A$, $P_B$ gets $P_T$'s public key $u_T$ to verify that $s_T$ is a valid signature of $P_T$ on $\iota_A$ and $\delta_A$. This is done via function *Verify*(·) defined in Section 3.1, i.e., $P_B$ confirms that $Verify(s_T, H(\iota_A \| \delta_A), u_T) = yes$. $P_B$ must also make sure that the information in header $\iota_A$ is valid to $P_A$, e.g., $P_A$'s identity and public key match those in $\iota_A$, and $C_A$ is not expired.

After the successful validation of $C_A$, $P_B$ proceeds to verify $K_A$ by performing the following calculation with the data items in $K_A$ and $C_A$ together with hashes $h_A$ and $d_A$:

$$\beta'_A = (g^{e_A} \times (u_A{}^{\eta_A} \times y_A \times \delta_A)^{-1}) \bmod p, \text{ and } \eta'_A = H(h_A \| d_A \| \beta'_A \| y_A \| w_A).$$

$P_B$ then compares $\eta'_A$ with $\eta_A$ in $K_A$. If $\eta_A = \eta'_A$, $K_A$ is valid. This is due to the following relationship:

$$\begin{aligned}
\beta'_A &= (g^{e_A} \times (u_A{}^{\eta_A} \times y_A \times \delta_A)^{-1}) \bmod p \\
&= (g^{\gamma_A + \kappa_A} \times (g^{\eta_A \times r_A} \times g^{x_A} \times g^{k_A})^{-1}) \bmod p \\
&= (g^{\eta_A \times r_A + \alpha_A + k_A + x_A} \times g^{-(\eta_A \times r_A + k_A + x_A)}) \bmod p \\
&= g^{a_A} \bmod p \\
&= \beta_A.
\end{aligned}$$

The above verification of both $C_A$ and $K_A$ is signified by the following function in order to simplify the presentation of the exchange protocol to be proposed in Section 5:

$$o = Check(h_A, d_A, K_A, C_A).$$

The function has the result $o = yes$ if both $K_A$ and $C_A$ are valid in relation to $h_A$ and $d_A$, and $o = no$ otherwise.

The successful validation of $K_A$ and $C_A$ together with other checks to be detailed in Section 5 convinces $P_B$ that it is safe to release its agreed signature $S_B$ to $A_A$. In return, $A_A$ should send its proxy signature $S_A$ yielded in Section 4.3 to $P_B$. To verify $S_A$, $P_B$ computes public proxy signature key $\mu_A = (u_A{}^{\eta_A} \times \beta'_A) \bmod p$ as defined in Section 4.3. $P_B$ then verifies $S_A$ via function $Verify(\cdot)$ given in Section 3.1, namely, checking that $Verify(S_A, H(h_A \| d_A), \mu_A) = yes$. If $S_A$ is valid, the exchange is completed successfully with no need for any involvement of $P_T$.

In case $P_B$ fails to receive correct $S_A$ after handing over $S_B$ to $A_A$, $P_B$ can request $P_T$ to recover a valid proxy signature $\varsigma_A$ of $P_A$ on $h_A$ and $d_A$ from $K_A$, which will be detailed in the next sub-section.

### 4.5. Proxy Signature Recovery

A signature recovery is needed only when $P_B$ is unable to obtain a valid proxy signature of $P_A$ from $A_A$ after releasing its own signature to $A_A$. The recovery is intended to assure the fair completion of the exchange in abnormal circumstances. To achieve this, $P_T$ needs to decrypt encrypted proxy signature key $\tau_A$ using long-term key $k_A$ shared with $P_A$, and then generates a proxy signature of $P_A$ with $\tau_A$ for $P_B$.

To request $P_T$ for a signature recovery, $P_B$ should pass $h_A$, $d_A$, $K_A$ and $C_A$ to $P_T$. Upon receipt of the request, $P_T$ examines the validity of $K_A$ and $C_A$ in the same way used by $P_B$ in Section 4.4, i.e., confirming that $Check(h_A, d_A, K_A, C_A) = yes$. If the validation is successful and all the other conditions to be stated in Section 5 are satisfied, $P_T$ applies its private key $r_T$ and header $\iota_A$ in received $C_A$ to compute:

$$k_A = H(r_T \| \iota_A), \text{ and } \tau_A = (e_A - k_A) \bmod q.$$

Here, $e_A = (\tau_A + k_A) \bmod q$ was defined in Section 4.3.

$P_T$ then uses $\tau_A$ as a private proxy signature key to generate a proxy signature $\varsigma_A$ on $h_A$ and $d_A$ for $P_B$ on $P_A$'s behalf using the agreed signature scheme $Sign(\cdot)$ defined in Section 3.1, i.e. $\varsigma_A = Sign(H(h_A \| d_A), \tau_A)$.

$P_B$ can validate $\varsigma_A$ by forming the corresponding public proxy signature key $\upsilon_A = (u_A{}^{\eta_A} \times \beta'_A \times y_A) \bmod p$, as defined in Section 4.3, from its possessed items to confirm that $Verify(\varsigma_A, H(h_A \| d_A), \upsilon_A) = yes$.

As mentioned in Section 4.2, warrant $w_A$ in $K_A$ specifies the acceptable forms of $P_A$'s proxy signatures. Now we can explicitly state that these forms are restricted to those of $S_A$ and $\varsigma_A$. More specifically, for the given exchange, session and agreed document hashes $h_A$ and $d_A$ must appear in both private proxy signature key $\gamma_A$ (or $\tau_A$) and proxy signature $S_A$ (or $\varsigma_A$) in order for $P_A$ to accept $S_A$ (or $\varsigma_A$) as its valid proxy signature. In other words, any

signature, which does not meet this restriction, is an invalid proxy signature of $P_A$. This effectively turns $\gamma_A$ and $\tau_A$ into the one-time keys for the specified exchange, which were discussed in Section 4.1. This is because $A_A$ can neither yield $\gamma_A$ by itself nor alter it without invalidating it, as introduced in Sections 4.3 and 4.4. Additionally, applying $\gamma_A$ to sign a different hash $h'_A$ ($\neq h_A$) or $d'_A$ ($\neq d_A$) from the one in $\gamma_A$ would violate the above restriction so that the proxy signature produced is invalid to $P_A$. Hence $A_A$ can only use $\gamma_A$ to produce $S_A$. The same discussion is also applicable to $\tau_A$ and $\varsigma_A$ as $\tau_A$ is formed from $\gamma_A$ (see Section 4.3).

Additionally, $P_A$ is able to distinguish a proxy signature yielded by $A_A$ from that by $P_T$, because they use different private proxy signature keys for the signature generations and neither of $A_A$ and $P_T$ knows the other's key. This is a useful feature, which allows $P_A$ to trace the producer of a proxy signature when resolving a dispute about its validity.

It is worth emphasising that key $k_A$ shared between $P_A$ and $P_T$ is used only for the encryption and decryption of private proxy signature key $\tau_A$. $k_A$ is not a valid key for the generation of $P_A$'s signatures, namely, $P_T$ cannot employ $k_A$ to produce any valid signature of $P_A$.

Since $P_B$ only receives signature $\varsigma_A$ from $P_T$, $P_B$ is unable to derive key $\tau_A$ from $\varsigma_A$. This implies that $P_B$ cannot obtain any of the other keys $\gamma_A$, $\kappa_A$ and $k_A$ either. The detailed analysis of this claim will be presented in Section 6. Therefore, $\kappa_A$ is secure and reusable for further exchanges.

## 5. Fair Signature Exchange Protocol

Based on the verifiable proxy encryption approach proposed in Section 4, we now present a protocol for Proxy-Led Efficient Autonomous Signature Exchange (PLEASE) in u-commerce systems. It offers a series of prescribed message transmissions for $A_A$, $P_B$ and $P_T$ to follow in order to complete an agreed exchange fairly. The protocol is divided into two sub-protocols. One is for handing a normal case of exchange without signature recovery, which is denoted as PLEASE-E. The other is for processing signature recovery, which is represented as PLEASE-R, in case the normal exchange fails to complete properly. These two sub-protocols are illustrated in Fig. 2 and 3 respectively. They are based on the assumption that there is a secure channel between any two communicating parties involved, e.g., the communications can be protected using SSL/TLS [9].

The first sub-protocol PLEASE-E shown in Fig. 2 specifies a sequence of operations executed by each of proxy exchanger $A_A$ and party $P_B$. The sub-protocol is initiated by $A_A$. To form the first message, $A_A$ performs its top

group of operations in Fig. 2. Specifically, $A_A$ gathers the relevant information about the current exchange to define a session header $\mu_A$. It serves to indicate the identities of $P_A$ and $P_B$ involved in the exchange, their agreed document hashes $d_A = H(D_A)$ and $d_B = H(D_B)$ to be signed using the agreed signature scheme, a timestamp, and a completion deadline to prevent a party from purposely delaying the completion. $A_A$ also gets hold of the assigned key certificate, warrant and proxy encryption key introduced in Section 4.2. Based on these items, $A_A$ requests the designated signers to jointly produce a Schnorr signature $(\gamma_A, \eta_A)$ on hashes $h_A$ and $d_A$, and then encrypts $\gamma_A$ with proxy encryption key $\kappa_A$ for the formation of $K_A$ as the verifiable encryption of proxy signature key $\tau_A$ assigned to TTP $P_T$, as defined in Section 4.3. $A_A$ then sends $\mu_A$, $K_A$ and $C_A$ as its first message to $P_B$.

Note that there is no need to have an additional signature for the integrity protection of $A_A$'s message. This is because $\mu_A$, $K_A$ and $C_A$ in the message are linked together, as $C_A$ is associated with $\kappa_A$ used for the encryption $e_A$ of $\gamma_A$ in signature $(\gamma_A, \eta_A)$ on $\mu_A$ as well as parameter $y_A$ and warrant $w_A$ in $K_A$.

Moreover, the authenticity of the message is established by $P_B$ via the verification of $K_A$. As shown in Section 4.4, the verification resembles to that of original signature $(\gamma_A, \eta_A)$ without revealing $\gamma_A$ to $P_B$. This verifiable but hidden signature serves to assure $P_B$ that $P_T$ has been given a permission to recover a proxy signature of $P_A$ for $P_B$, provided that the following conditions are met:

(a) $P_B$ submits a valid signature $S_B$ on session hash $h_A = H(\mu_A)$ and document hash $d_B$ to $P_T$, and

(b) the exchange satisfies all the conditions specified in $\mu_A$ and $w_A$.

Upon the reception of $A_A$'s message, $P_B$ begins the execution of its top group of operations in Fig. 2. These include the validation of $C_A$ and $K_A$ via function $Check(\cdot)$ defined in Section 4.4 and the examination of the conditions in $\mu_A$ and $w_A$ to ensure the authenticity and integrity of $A_A$'s message and the satisfaction of condition (b) stated above. A failure of the verification or examination leads to protocol termination as the message is invalid for the current exchange. In this case, neither of $A_A$ and $P_B$ has disclosed its signature to the other. Otherwise, the valid message convinces $P_B$ that $P_T$ can recover a valid proxy signature of $P_A$ from the message when $P_B$ meets the above condition (a). With this assurance, $P_B$ generates its signature $S_B$ using function $Sign(\cdot)$ defined in Section 3.1, and then sends it to $A_A$ as the second message in Fig. 2. Here, $h_A$ in the message is used as an identifier to indicate which exchange session the message belongs to, as $A_A$ may run multiple exchange sessions concurrently.

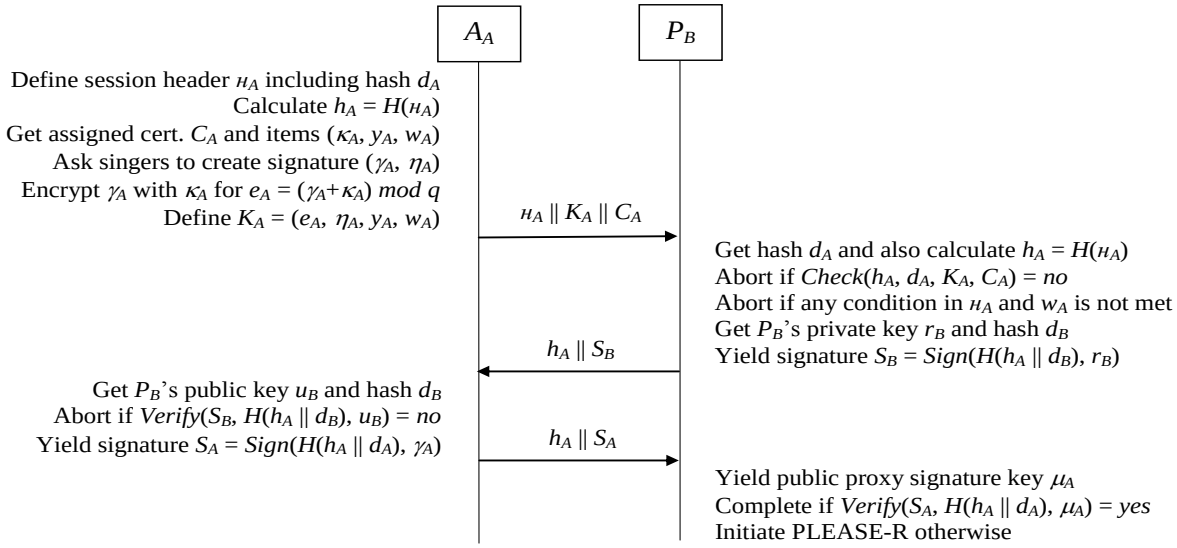


Fig. 2. Signature exchange sub-protocol PLEASE-E.

In response to the arrival of $P_B$'s message, $A_A$ performs its bottom group of operations in Fig. 2. If received signature $S_B$ is proved to be invalid by function $Verify(\cdot)$ specified in Section 3.1, $A_A$ simply terminates the protocol run because $S_B$ is not the signature that $A_A$ can accept. In this situation, neither $A_A$ nor $P_B$ has released its valid signature to the other, although $P_B$ has got hold of encrypted proxy signature key $\tau_A$ from $A_A$ via the first message. Otherwise, $A_A$ proceeds to yield a proxy signature $S_A$ with its private proxy signature key $\gamma_A$, and then transfers $S_A$ to $P_B$ as the third message in Fig. 2 to complete its exchange process.

Having received $S_A$ from $A_A$, $P_B$ executes its bottom group of operations in Fig. 2. If the validity of $S_A$ is confirmed with public proxy signature key $\mu_A$ defined in Section 4.3, the exchange has been completed successfully. Otherwise, $P_B$ cannot accept $S_A$ as a valid signature of $P_A$. In this case, $P_B$ has already handed over its valid signature $S_B$ to $A_A$ but failed to get a valid signature from $A_A$. Thus it is essential for $P_B$ to activate sub-protocol PLEASE-R in Fig. 3 to request $P_T$ for the recovery of a valid signature of $P_A$.

To run PLEASE-R, $P_B$ gathers its signature $S_B$ and $A_A$'s first message received via PLEASE-E and then sends them to $P_T$ as the first message in Fig. 3.

Upon the receipt of the message from $P_B$, $P_T$ executes its sequence of operations defined in Fig. 3. These include extracting relevant information from the message, e.g. document hashes $d_A$ and $d_B$ in header $\varkappa_A$, and verifying the validity of $S_B$ and $A_A$'s message in the ways discussed for the first two messages of PLEASE-E in Fig. 2. $P_T$ accepts $P_B$'s signature recovery request only if all the verifications are passed successfully. To proceed for the recovery of a signature of $P_A$ for $P_B$, $P_T$ decrypts encrypted private proxy signature key $\tau_A$ using the information in $A_A$'s

message forwarded from $P_B$, and applies decrypted $\tau_A$ to create a proxy signature $\varsigma_A$ on $P_A$'s behalf, as presented in Section 4.5. $P_T$ then sends $\varsigma_A$ to $P_B$, and forwards $S_B$ to $A_A$, as the second and third messages in Fig.3 respectively.

Similar to the last two messages of PLEASE-E in Fig. 2, each of $P_B$ and $A_A$ validates the signature received from $P_T$. There is no need for $A_A$ to verify $S_B$ again if $A_A$ already got correct $S_B$ via PLEASE-E. When both $P_B$ and $A_A$ have obtained their expected signatures, the exchange is completed.

Note that $P_B$ or $A_A$ may not receive its message from $P_T$ correctly due to communication or system failures. In this case, $P_B$ or $A_A$ can request $P_T$ for the message re-transmission. Hence, $P_T$ should keep $\varsigma_A$ and $S_B$ for a specified period to serve such a re-transmission purpose. Of course, $P_B$ can re-run PLEASE-R if necessary.

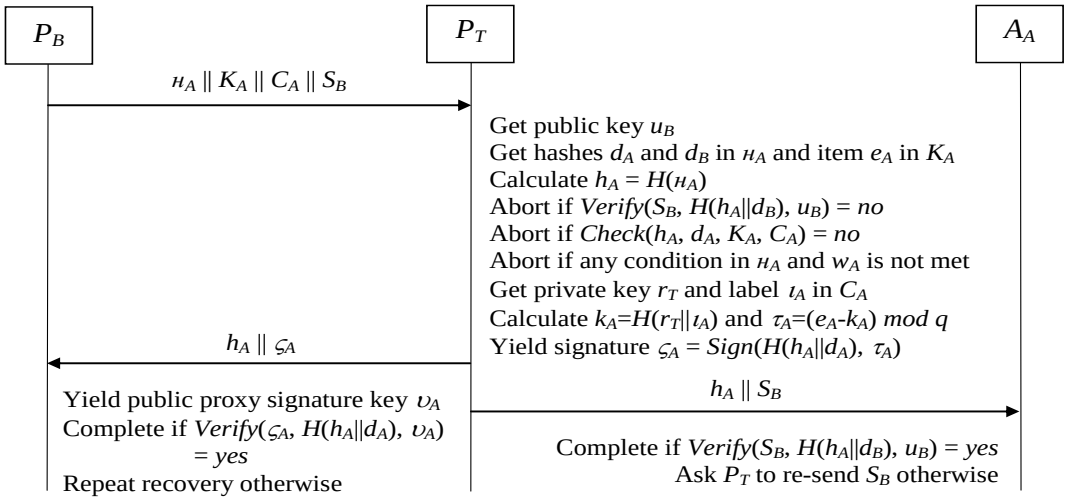


Fig. 3. Signature recovery sub-protocol PLEASE-R.

## 6. Security Analysis

We now evaluate the security assurance of the approach to the verifiable proxy encryption of proxy signature keys proposed in Section 4. This includes analysing the security strength of the approach, its signature recoverability and exchange fairness. We begin with the following claim about the approach's security strength:

**Claim 1:** *The approach to the verifiable encryption $K_A$ of private proxy signature key $\tau_A$ presented in Section 4.3 is as secure as the Schnorr signature scheme introduced in Section 3.2.*

**Analysis:** As shown in Section 4.3, the encrypted item $e_A$ in $K_A$ can be expressed as:

$$e_A = (\gamma_A + \kappa_A)\ mod\ q = (\gamma_A + x_A + k_A)\ mod\ q = (k_A + \tau_A)\ mod\ q.$$

Here, $\gamma_A$ and $\tau_A$ are the two private proxy signature keys created for $A_A$ and $P_T$, respectively. $k_A$ is the key shared

between $P_A$ and $P_T$. $\kappa_A$ is the symmetric proxy encryption key produced from a random number $x_A$ ($< q$) and key $k_A$ for $A_A$.

Moreover, an original Schnorr signature ($s_A$, $m_A$) on a document hash $d'_A$ is yielded with a random number $a_A < q$ and $P_A$'s private key $r_A$ in the following way introduced in Section 3.2:

$$b_A = g^{a_A} \bmod p,\ m_A = H(d'_A \,||\, b_A), \text{ and } s_A = (m_A \times r_A + a_A) \bmod q.$$

By comparing $e_A = (k_A + \tau_A) \bmod q$ with $s_A$, it is evident that the form of $e_A$ is a special case of the form of $s_A$ with $m_A = 1$, where $k_A$ corresponds to $r_A$, which both remain unchanged, and $\tau_A$ matches $a_A$, which both vary, for different encryptions and signatures, respectively. Note that the variation of $\tau_A = (\gamma_A + x_A) \bmod q$ for different exchanges is owing to the inclusion of a new random number $\alpha_A < q$ in the formation of each $\gamma_A = (\eta_A \times r_A + \alpha_A) \bmod q$ as shown in Section 4.3. Thus, deriving $k_A$ or $\tau_A$ from $e_A$ is as hard as obtaining $r_A$ or $a_A$ from $s_A$.

Similarly, $k_A$ embedded in proxy encryption key $\kappa_A = (k_A + x_A) \bmod q$ is as secure as $r_A$ in $s_A$, and also $\kappa_A$ is only known by $P_A$ and $A_A$. Note that $A_A$ cannot change $\kappa_A$ without invalidating it. This is because $k_A$ in $\kappa_A$ is attached to $\delta_A = g^{k_A} \bmod p$ in certificate $C_A = (\iota_A, \delta_A, s_T)$ issued by $P_T$ in Section 4.2, and $x_A$ is fixed via $y_A = g^{x_A} \bmod p$ in hash $\eta_A$ in Schnorr signature ($\gamma_A$, $\eta_A$) produced jointly by the appointed signers for $A_A$ in Section 4.3. For example, if $A_A$ is compromised, it may attempt to maliciously alter $\kappa_A$ and $y_A$ to get different values $\kappa'_A = (\kappa_A + \sigma) \bmod q = (k_A + x_A + \sigma) \bmod q$ and $y'_A = (y_A \times g^{\sigma}) \bmod p = g^{x_A+\sigma} \bmod p$ with a random number $\sigma < q$, and apply $\kappa'_A$ and $y'_A$ to the formation of verifiably encrypted proxy signature key $K'_A$, which is then sent to $P_B$. Although the relationship between $\kappa'_A$ and $y'_A$ appears to be right, the verification of $K'_A$ performed by $P_B$ in Section 4.4 will fail due to $\eta_A \neq H(h_A \,||\, d_A \,||\, \beta'_A \,||\, y'_A \,||\, w_A)$, where $\eta_A = H(h_A \,||\, d_A \,||\, \beta_A \,||\, y_A \,||\, w_A)$, $\beta_A = \beta'_A$ but $y_A \neq y'_A$. Consequently, $P_B$ rejects $K'_A$, so $A_A$ gains no benefit from its misbehaviour.

Note that $A_A$ itself is unable to generate any valid signature of $P_A$, and no adversary could compromise enough signers to obtain a valid signature of $P_A$ illegitimately, as assumed in Section 2.1. This implies that $A_A$ cannot get enough signers to sign altered value $y'_A$ illicitly.

Additionally, since $\gamma_A$ in signature ($\gamma_A$, $\eta_A$) is directly used by $A_A$ as a private proxy signature key for the generation of proxy signature $S_A$, $\gamma_A$ inherits the security of the Schnorr signature scheme. This also applies to private proxy signature key $\tau_A = (\gamma_A + x_A) \bmod q$ passed to $P_T$ using $e_A$, because the only difference from $\gamma_A$ is that $\tau_A$ contains extra random number $x_A$ with $y_A = g^{x_A} \bmod p$ included in $\eta_A$, namely, ($\tau_A$, $\eta_A$) can also be seen as a Schnorr

signature of $P_A$.

Recall that $A_A$ is only permitted to hold $\kappa_A$ and $\gamma_A$. As analysed above, deriving $k_A$ or $x_A$ from $\kappa_A$ by $A_A$ is as hard as breaking the security of the Schnorr scheme. Thus, it is hard for $A_A$ to construct $\tau_A$ from $\kappa_A$ and $\gamma_A$. On the other hand, $P_T$ is only allowed to get $\tau_A$ and $k_A$, so it cannot compute $\gamma_A$ without knowing $x_A$ or $\kappa_A$. Moreover, since each of $\gamma_A$ and $\tau_A$ together with $\eta_A$ is a Schnorr signature of $P_A$ on given session and document hashes $h_A$ and $d_A$, $\gamma_A$ and $\tau_A$ cannot be altered without invalidating them. They are acceptable by $P_A$ if and only if they are used to sign both $h_A$ and $d_A$ that must also appear in $\eta_A$, as discussed about warrant $w_A$ in Section 4.5. In other words, $\gamma_A$ and $\tau_A$ are invalid for signing any hashes other than $h_A$ and $d_A$. Here, no adversary could illegitimately obtain a signature of $P_A$ from its appointed signers, as assumed in Section 2.1. These features enable $P_A$ to distinguish proxy signatures produced by $A_A$ from those by $P_T$.

It is worth pointing out that the security of proxy signatures $S_A$ and $\varsigma_A$ signed with keys $\gamma_A$ and $\tau_A$ respectively as illustrated in Fig. 2 and 3 relies completely on the agreed signature scheme signified by functions *Sign*(·) and *Verify*(·) specified in Section 3.1, because the work proposed in this paper makes no change to the scheme.

There is also a possibility that party $P_B$ tries to illicitly obtain key $k_A$ from certificate $C_A = (\iota_A, \delta_A, s_T)$ defined in Section 4.2. As $k_A = H(r_T \parallel \iota_A)$, it is hard for $P_B$ to directly compute $k_A$ without knowing $P_T$'s private key $r_T$. Moreover, recovering $k_A$ from $\delta_A = g^{k_A} \bmod p$ (i.e., $k_A$ and $\delta_A$ actually form a pair of private and public keys) is equivalent to deriving a private key from its associated public key, under the Schnorr signature scheme. This is hard to achieve. Otherwise, the Schnorr scheme would not be secure.

Furthermore, any other party ($\notin \{P_A, P_B, P_T\}$) cannot even get $K_A$ and $C_A$, as they are sent from $A_A$ to $P_B$ via sub-protocol PLEASE-E, or forwarded from $P_B$ to $P_T$ via PLEASE-R, over a secure channel, as stated in Section 5.

It is evident from the above analysis that the security of encryption $K_A$ is as strong as that of the Schnorr signature scheme. □

We now proceed to confirm the recoverability of the proposed approach with the following claim:

**Claim 2:** *Given valid certificate $C_A$ and encryption $K_A$, only $P_T$ apart from $P_A$ can certainly recover correct private proxy signature key $\tau_A$ for the generation of a required proxy signature of $P_A$ without any involvement of $A_A$ and $P_A$.*

**Analysis:** As presented in Section 4.4, valid $C_A = (\iota_A, \delta_A, s_T)$ means that it is indeed $P_A$'s certificate issued by $P_T$, namely, $P_T$ can recover shared key $k_A$ from $C_A$, which is guaranteed to be the exponent in $\delta_A = g^{k_A} \bmod p$. Moreover, valid $K_A = (e_A, \eta_A, y_A, w_A)$ indicates that it meets the condition $\eta_A = H(h_A \| d_A \| \beta'_A \| y_A \| w_A)$ with $\beta'_A = (g^{e_A} \times (u_A{}^{\eta_A} \times y_A \times \delta_A)^{-1}) \bmod p$, where $u_A = g^{r_A} \bmod p$ is $P_A$'s public key associated with its private key $r_A$. This verification resembles to that of Schnorr signature $(\gamma_A, \eta_A)$ created in Section 4.3, with extra numbers $y_A$ and $\delta_A$ fixed by $\eta_A$ and $C_A$, respectively. The validity of $K_A$ implies that $\beta'_A$ is equal to $\beta_A = g^{\alpha_A} \bmod p$ included in $\eta_A$.

By re-arranging $\beta_A = (g^{e_A} \times (u_A{}^{\eta_A} \times y_A \times \delta_A)^{-1}) \bmod p$, we have:

$$(u_A{}^{\eta_A} \times \beta_A \times y_A) \bmod p = (g^{e_A} \times \delta_A{}^{-1}) \bmod p.$$

This leads to:

$$g^{\eta_A \times r_A + \alpha_A + x_A} \bmod p = g^{e_A - k_A} \bmod p.$$

Since private proxy signature key $\tau_A$ was defined as $\tau_A = (\gamma_A + x_A) \bmod q = (\eta_A \times r_A + \alpha_A + x_A) \bmod q$ in Section 4.3, the above equation can be expressed as:

$$g^{\tau_A} \bmod p = g^{e_A - k_A} \bmod p.$$

This is equivalent to:

$$\tau_A = (e_A - k_A) \bmod q.$$

The above result means that given valid $K_A$ and $C_A$, $P_T$ can certainly regain $k_A$ from $C_A$ as discussed in Section 4.5, and then use $e_A$ in $K_A$ to calculate $(e_A - k_A) \bmod q$ for $\tau_A$. Recovered key $\tau_A$ enables $P_T$ to create a required proxy signature on $h_A$ and $d_A$ for $P_B$, provided that all the conditions discussed in Section 5 are satisfied. Clearly, the recovery of $\tau_A$ by $P_T$ does not need any involvement of $A_A$ and $P_A$.

It is clear from the above analysis that $P_T$ can certainly obtain a valid proxy signature of $P_A$ from correct $K_A$ and $C_A$ independently of $A_A$ and $P_A$. □

The two claims analysed above can be applied to support the following claim:

**Claim 3:** *The protocol PLEASE presented in Section 5 can satisfy the fairness requirement stated in Section 2.2.*

**Analysis:** To confirm the validity of this claim, we need to demonstrate two cases of signature acquirement according to the fairness requirement. The first case is to prove that if $A_A$ has obtained the valid signature $S_B$ of

$P_B$, then $P_B$ has gained or can get a valid proxy signature of $P_A$. The second case swaps the positions of $A_A$ and $P_B$, i.e., showing that if $P_B$ has got a valid proxy signature of $P_A$, $A_A$ has obtained or can get the valid signature $S_B$ of $P_B$. Collectively, the two cases confirm that either each of $A_A$ and $P_B$ or neither of them has gained the other's valid signature at the end of the exchange, which is required for the exchange fairness.

We start with the analysis of the first case. This means that $A_A$ is supposed to have obtained valid signature $S_B$ from $P_B$ or through $P_T$, as there is no other way for $A_A$ to acquire $S_B$, assuming that $P_B$'s private signature key is not compromised. We then must show that $P_B$ has gained or can get a valid proxy signature of $P_A$. In this case, the attainment of $S_B$ by $A_A$ indicates that $P_B$ has certainly received the correct items from $A_A$ in the first message of sub-protocol PLEASE-E in Fig. 2. Otherwise, $P_B$ would not have released $S_B$ to $A_A$. According to Claim 2 discussed earlier, the possession of $A_A$'s correct message guarantees $P_B$ that $P_T$ can certainly recover $P_A$'s proxy signature $\varsigma_A$ for $P_B$, when $P_B$ makes a valid signature recovery request to $P_T$ as discussed in Section 5. More specifically, $P_B$ could have received $P_A$'s proxy signature $S_A$ directly from $A_A$ in the third message of PLEASE-E, or $P_B$ can definitely obtain $\varsigma_A$ indirectly from $P_T$ by activating PLEASE-R in Fig. 3. Hence, $P_B$ can certainly get one of $S_A$ and $\varsigma_A$, which are equally valid for given session and document hashes $h_A$ and $d_A$.

For the second case stated earlier, suppose that $P_B$ have obtained a valid proxy signature of $P_A$. In other words, $P_B$ has received either $S_A$ from $A_A$ in the third message of PLEASE-E or $\varsigma_A$ from $P_T$ in the second message of PLEASE-R, as there is no illicit way for $P_B$ to obtain the signature according to the analysis of Claim 1. We now show that $A_A$ has gained or can get $P_B$'s signature $S_B$. Since $P_B$ obtained either $S_A$ via PLEASE-E or $\varsigma_A$ through PLEASE-R, $A_A$ is able to get $S_B$ via one of these two sub-protocols as well. If $P_B$ gained $S_A$, $A_A$ had certainly received $S_B$ from $P_B$ via the second message of PLEASE-E. Otherwise, $A_A$ would not have released $S_A$ to $P_B$ through the third message of PLEASE-E. If $P_B$ got $\varsigma_A$, $A_A$ should have received $S_B$ from $P_T$, because $P_T$ passed $\varsigma_A$ to $P_B$, while forwarding $S_B$ to $A_A$, via PLEASE-R. $A_A$ can also request $P_T$ for the re-transmission of $S_B$ in case $A_A$ failed to receive $S_B$ from $P_T$, as discussed in Section 5. Thus, $A_A$ can obtain $S_B$.

Additionally, $P_B$ may not follow PLEASE-E properly after the reception of the first message from $A_A$. Instead $P_B$ uses the message to activate PLEASE-R for signature recovery. However, $P_B$ cannot get $P_A$'s signature unfairly in this case. According to PLEASE-R defined in Fig. 3, $P_B$ has to provide its valid signature for the signature recovery, and $P_T$ recovers $P_A$'s signature for $P_B$ while forwarding $P_B$'s signature to $A_A$, in order to ensure that the signature recovery is fair to both $P_A$ and $P_B$.

The above analysis demonstrates that either each of $A_A$ (or $P_A$) and $P_B$, or neither of them, can get the expected signature from the other at the end of the exchange, namely, protocol PLEASE satisfies the fairness requirement stated in Section 2.2. □

## 7. Comparison with Related Work

The issue of fair signature exchange between two distrusted parties over networks has been well studied, e.g. [2-5, 7, 14, 18-20, 22-25]. As pointed out in Section 1, the existing approaches to such exchange can be classified into the public-key and symmetric-key based categories in terms of the types of verifiable signature encryption employed to achieve the exchange fairness. A common way for public-key based verifiable signature encryption (e.g. [3]) is to require that one party $P_A$ apply a public key of a TTP $P_T$ to encrypt its signature $S_A$, and the other party $P_B$ can verify that the encryption contains correct $S_A$ and $P_T$ has the associated private key to decrypt the encryption for the recovery of $S_A$. Evidently, such public-key based encryption can be delegated by $P_A$ to its agent(s) because the agent only needs to know the public key to perform the encryption. While the approaches in this public-key based category offer the benefits of assuring the exchange fairness and enabling the signature exchange delegation, they are in general mathematically complex and computationally expensive to operate.

On the other hand, the symmetric-key based category of approaches to verifiable signature encryption has also been proposed (e.g. [19, 25]). They exhibit better simplicity and efficiency in comparison with any public-key based solutions. For instance, the modular exponentiations (the most expensive computations) needed by the symmetric-key based approach for a verifiable RSA [15] signature encryption and its verification given in [25] are about one quarter of those needed by a well-known public-key based method presented in [3].

However, as pointed out in Section 1, the existing symmetric-key based approaches do not normally offer the capability of delegating the task of verifiable signature encryption to a selected agent, which are ineffective for u-commerce systems. A recently proposed approach [20] attempts to address the problem by offering such delegation capability for signature exchange. Nevertheless, this approach utilises two separate signatures for one verifiable encryption. As discussed in Section 2.1, the joint signature generation is important for u-commerce systems to achieve better security and reliability, but it requires much more computation and communication than the signature generation only by one signer. Hence, the use of two signatures instead of one for a single verifiable encryption significantly weakens the efficiency of the approach in u-commerce settings.

The above discussion on the related work highlights that the public-key based approaches to fair signature

exchange are usable for u-commerce systems but inefficient, and that the existing symmetric-key based approaches are ineffective or inefficient in u-commerce settings.

Comparing with the related work, the new protocol PLEASE presented in this paper not only achieves the exchange fairness as analysed in the previous section but also shows the following novel characteristics:

(a) The approach proposed in Section 4 is the first one offering verifiable symmetric proxy encryption based on the Schnorr signature scheme.

(b) The proxy encryption of private proxy signature keys instead of proxy signatures, which differs from existing approaches, helps to achieve a better level of simplicity, efficiency and flexibility. Our approach is simpler and more efficient because it is built solely on the Schnorr signature scheme itself, which will be justified further later. It also uses only one signature (i.e. ($\gamma_A$, $\eta_A$)) for each verifiable encryption, which overcomes the weakened efficiency problem discussed earlier about the existing symmetric-key based approaches. These make our approach easier to implement on small resource-limited devices normally used in u-commerce systems. This advantage is important, particularly when the approach will be extended to allow joint proxy encryption by multiple agents for better security and reliability. The approach is also flexible as the private proxy signature keys proposed in Section 4 are applicable to the whole family of discrete logarithm based signature schemes for proxy signature generations [16]. In other words, the approach is suited to fair exchanges of any such signatures.

(c) The proxy encryption is supported by the issuance of a short-term purpose-restricted proxy encryption key $\kappa_A$ from a long-term key $k_A$ shared between $P_A$ and $P_T$ with a warrant $w_A$. This enables party $P_A$ to delegate the verifiable encryption to its exchanger $A_A$ while still keeping $k_A$ secure from $A_A$. Thus our approach resolves the encryption delegation problem, experienced by the existing symmetric-key based approaches, in a simple way.

(d) The proposed private proxy signature keys for proxy signature generations are also augmented with the one-time property, which permits each signature key to be valid only for one specific exchange signified by a pair of session and document hashes. This feature prevents the signature keys from being used for any illegitimate purposes.

To support the above claim on the better simplicity and efficiency of our approach, we compare it with the well-known public-key based approach to the verifiable encryption of Schnorr signatures, which was proposed by

Ateniese in [3] and could be applied to fair signature exchange in u-commerce. The simplicity of our approach lies in the fact that it is based solely on the Schnorr signature scheme itself, as analysed in Section 6. However, Ateniese's approach requires two additional techniques, the Naccache-Stern cryptosystem [11] for signature encryption and a proof-of-knowledge scheme for encrypted signature verification, so our approach is mathematically much simpler.

To compare the efficiency of the two approaches, we consider the generation, verification and recovery of a verifiably encrypted key or signature with regard to the number of the most expensive computations, i.e. modular exponentiations. As listed in Table 2, our approach needs 2, 2 and 3 modular exponentiations for the generation, verification and recovery of a verifiably encrypted signature key, respectively. Here, the generation and recovery include the production of proxy signatures $S_A$ and $\varsigma_A$ by $A_A$ and $P_T$, respectively. Also we only count one exponentiation for the joint generation of Schnorr signature ($\gamma_A$, $\eta_A$) in Section 4.3 as such joint signature generation should also be used by Ateniese's approach for u-commerce systems. Moreover, the creation of encryption key $\kappa_A$ in Section 4.2 is excluded because $\kappa_A$ is used for multiple exchanges, and the verification of certificate $C_A$ defined in Section 4.2 is not counted as Ateniese's approach needs a similar verification as well.

| **Number of Modular Exponentiations** | | | |
|---|---|---|---|
| **Approach** | **Generation** | **Verification** | **Recovery** |
| Ateniese's Approach | 5 | 5 | 4 |
| Our Approach | 2 | 2 | 3 |

Table 2: Computation comparison between our and Ateniese's approaches for the generation, verification and recovery of a verifiably encrypted key and signature.

In contrast, Ateniese's approach requires 5, 5 and 4 modular exponentiations for the generation, verification and recovery of a verifiably encrypted Schnorr signature, respectively. Since Ateniese did not define the recovery details in [3], we assume that the encrypted signature is decrypted first and the recovered signature is then verified, which is more efficient than verifying the encrypted signature and then decrypting it. Also we count only 2 modular exponentiations for the decryption as mentioned in [3].

Additionally, Table 3 provides a comparison between our and Ateniese's approaches in terms of the numbers of data items produced for a verifiably encrypted key and signature. Here, a *q*-size, *p*-size or hash-size data item means that its length is not longer than that of *q*, *p* or a hash, respectively, with one exception that one of the 2 *p*-

size data items for Ateniese's approach can be larger than $p$ due to the use of the Naccache-Stern cryptosystem for signature encryption [3]. Also, warrant $w_A$ specified in Section 4.2 is not counted as it should also be adopted by Ateniese's approach in u-commerce settings. Moreover, certificate $C_A$ is not considered because a similar certificate is used by Ateniese's approach as well. From Table 3, it is evident that our approach yields less data than Ateniese's approach.

| **Number of Data Items** | | | |
|---|---|---|---|
| **Approach** | ***q*-size** | ***p*-size** | **Hash-size** |
| Ateniese's Approach | 1 | 2 | 2 |
| Our Approach | 1 | 1 | 1 |

Table 3: Data comparison between our and Ateniese's approaches in terms of the numbers of data items generated for a verifiably encrypted key and signature.

The above comparisons clearly illustrate that our approach is much more efficient and simpler than Ateniese's approach. This indicates that our approach is much easier to implement and more efficient to run on small computing devices used in u-commerce systems.

## 8. Conclusions and Future Work

We have identified the shortcomings of the existing work on fair signature exchange in relation to the emerging exchange scenarios of u-commerce. This has motivated us to propose a novel approach to the verifiable symmetric proxy encryption of one-time proxy signature keys in Section 4 to rectify the identified weaknesses. Such proxy encryption enables the task of verifiable encryption to be delegated to a chosen agent without disclosing any long-term key to the agent, whilst retaining the simplicity and efficiency of the symmetric-key based encryption. The use of proxy signature keys instead of proxy signatures for their verifiable encryption makes the approach flexible for its application to the whole family of discrete logarithm based signature schemes with no need for any modification. Also the one-time property of the proxy signature keys prevents them from being abused for illegitimate purposes.

In addition, the proposed approach has been applied to the design of a new protocol suited to emerging autonomous fair signature exchange in u-commerce in Section 5. The aforementioned characteristics of the approach distinguish the protocol from other existing ones, and help to bridge a critical gap in the security

protection of u-commerce. The protocol analysis conducted in Section 6 has confirmed that it can assure the security and fairness of signature exchanges. The comparison with related work presented in Section 7 has justified that the new approach is much simpler and more efficient than the relevant existing ones.

For the future work, we intend to extend the approach for joint verifiable proxy encryption using multiple distributed agents for better security and reliability.

## Acknowledgement

We would like to thank the paper reviewers for their valuable suggestions, which have led to improvements to the paper.

## References

[1] N. Asokan, V. Shoup, M. Waidner, Asynchronous protocols for optimistic fair exchange, in: Proceedings of the IEEE Symposium on Security and Privacy, Oakland, USA, 1998, IEEE CS, pp. 86-100.

[2] N. Asokan, V. Shoup, M. Waidner, Optimistic fair exchange of digital signatures, IEEE Journal on Selected Areas in Communications 18 (4) (2000) 593-610.

[3] G. Ateniese, Verifiable encryption of digital signatures and applications, ACM Transactions on Information and Systems Security 7 (1) (2004) 1-20.

[4] F. Bao, R. Deng, W. Mao, Efficient and practical fair exchange protocols with off-line TTP, in: Proceedings of the IEEE Symposium on Security and Privacy, Oakland, USA, 1998, IEEE CS, pp. 77-85.

[5] J. Camenisch, V. Shoup, Practical verifiable encryption and decryption of discrete logarithms, in: Proceedings of CRYPTO 2003, California, USA, 2003, LNCS, vol. 2729, Springer, Berlin, pp. 126-144.

[6] T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, IEEE Transactions on Information Theory IT-31 (4) (1985) 469-72.

[7] P.D. Ezhilchelvan, S.K. Shrivastava, A family of trusted third party based fair-exchange protocols, IEEE Transactions on Dependable and Secure Computing 2 (4) (2005) 273-86.

[8] Q. Huang, G. Yang, D.S. Wong, W. Susilo, Ambiguous optimistic fair exchange, in: Proceedings of the ASIACRYPT 2008, Melbourne, Australia, 2008, Springer, Berlin, pp. 74-89.

[9] IETF, The Transport Layer Security (TLS) Protocol - Version 1.2, RFC 5246, 2008.

[10] M. Mambo, K. Usuda, E. Okamoto, Proxy signatures: delegation of the power to sign messages, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, E79-A (9) (1996)

1338-54.

[11] D. Naccache, J. Stern, A new public key cryptosystem based on higher residues, in: Proceedings of the 5th ACM Conference on Computer and Communications Security, San Francisco, USA, 1998, pp. 59-66.

[12] NIST, Digital Signature Standard, Federal Information Processing Standards Publication, vol. 186, 1994.

[13] NIST, Secure Hash Standard, Federal Information Processing Standards Publication, vol. 180-2, 2002.

[14] Y. Okada, Y. Manabe, T. Okamoto, An optimistic fair exchange protocol and its security in the universal composability framework, International Journal of Applied Cryptography 1 (1) (2008) 70-7.

[15] R.L. Rivest, A. Shamir, L.M. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM 26 (1) (1983) 96-9.

[16] B. Schneier, Applied Cryptography, John Wiley & Sons, 1996.

[17] C.P. Schnorr, Efficient signature generation by smart-cards, Journal of Cryptology 4 (3) (1991) 161-174.

[18] Z. Shao, Fair exchange protocol of signatures based on aggregate signatures, Computer Communications 31 (10) (2008) 1961-9.

[19] Q. Shi, N. Zhang, M. Merabti, Fair exchange of valuable information: A generalised framework, Journal of Computer and System Sciences 77 (2) (2011) 348-71.

[20] Q. Shi, N. Zhang, M. Merabti, R. Askwith, Achieving autonomous fair exchange in ubiquitous network settings, Journal of Network and Computer Applications 34 (2) (2011) 653-67.

[21] D.R. Stinson, R. Strobl, Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates, in: Proceedings of Australasian Conference on Information Security and Privacy (ACISP 2001), Sydney, Australia, 2001, LNCS, vol. 2119, Springer, Berlin, pp. 417-34.

[22] G. Wang, An abuse-free fair contract-signing protocol based on the RSA signature, IEEE Transactions on Information Forensics and Security 5 (1) (2010) 158-68.

[23] D.H. Yum, P.J. Lee, Efficient fair exchange from identity-based signature, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E91-A (1) (2008) 119-26.

[24] J. Zhang, C. Liu, Y. Yang, An efficient secure proxy verifiably encrypted signature scheme, Journal of Network and Computer Applications 33 (1) (2010) 29-34.

[25] N. Zhang, Q. Shi, A. Nenadic, M. Merabti, R. Askwith, Efficient fair exchange based on misbehaviour penalisation, IEE Proceedings – Communications 152 (3) (2005) 257-61.

## Author Biographies

**Q. Shi** is a Professor in Computer Security in the School of Computing & Mathematical Sciences at Liverpool John Moores University in the UK. He received his PhD in Computing from the Dalian University of Technology, P.R. China. Prior to joining the Liverpool John Moores University, he worked as a Research Associate for the Department of Computer Science at the University of York in the UK. His research interests include security protocol design, ubiquitous computing security, formal models, sensor network security, computer forensics, and intrusion detection. He has published extensively, and is supervising several projects, in these research areas.

**N. Zhang** is a Senior Lecturer in the Department of Computer Science, University of Manchester, UK. She received her PhD in Electronic Engineering from the University of Kent at Canterbury, UK. Her research interests include information and e-commerce security, computer networks, and mobile computing. She is supervising a number of research projects on these subjects, which are funded by various funding sources. She is also actively engaging in other academic activities including serving as a journal editor, a conference chair, and a member of journal editorial boards and conference program committees for a number of international journals and conferences.

**M. Merabti** is a Professor in the School of Computing & Mathematical Sciences at Liverpool John Moores University in the UK. He is a graduate of Lancaster University in the UK. His research interests include computer security, distributed multimedia systems, computer networks, and mobile computing. Prof. Merabti is widely published in these areas and has a number of government and industry supported research projects. He is a program chair for many international conferences and also serves as an editor and an editorial board member for a number of international journals.