Identity-Based Optimistic Fair Exchange with Transparent Signature Recovery

Shahrokh Saeednia, Olivier Markowitch and Yves Roggeman
Université Libre de Bruxelles, {saeednia,omarkow,yrogge}@ulb.ac.be

Abstract

In this paper, we propose a new practical fair exchange protocol allowing the exchange of an electronic item against a signature. The protocol is based on the Guillou-Quisquater scheme and assumes the existence of a trusted third party that is involved in the protocol only in the setup phase and when one of the parties does not follow the protocol or some technical problems occur during the execution of the protocol. The interesting feature of the protocol is the low communication and computational costs required by the parties. Moreover, in case of problems during the main protocol, the trusted third party acts transparently.

1 Introduction

Applications implying the exchange of information between users in a fair way, become more frequent. Payment systems, certified mail and contract signing are examples. An exchange protocol is said to be fair if after the execution of the protocol, either all the involved parties obtain their expected information or none of the information to be exchanged against the missing information are received. In this paper, we focus on the problem of the fair exchange of an electronic item, issued by a *provider*, against a digital signature considered as an acknowledgement of receipt of the item, issued by a *client*.

The previous major works about fair exchange assume the existence of a TTP in the protocol. Its role consists to prevent and/or resolve the problems that may occur between the communicating parties. In order to reduce the computation and communication overheads implied by the use of the TTP, Asokan and Micali proposed independently that the TTP has not to be involved in the transactions when the parties behave correctly and when the network functions, but to invoke the TTP to complete the protocol in case of problems. Protocols with such a TTP are said *optimistic* and the TTP is said to be *offline*. In the classical optimistic exchange protocols, in case of problems, the TTP forwards the item awaited by the client and can either make the client's signature available to the provider or give its own signature as an affidavit that has the same legal value than the client's signature. Our aim is to design a protocol, where the TTP is *invisible* or *transparent*. This means that at the end of the protocol, by only looking at the produced signatures, it is impossible to distinguish between the correct execution of the protocol and the case where the TTP completed it. As the intervention of the TTP can be due to a network failure rather than a cheating party, transparent TTPs are very useful in the context of electronic commerce, in order to avoid confusing reputations for both clients and providers. The idea of an invisible TTP was first presented by Micali. Later, Asokan et al. [1] and Boyd and Foo [2] proposed fair exchange protocols with transparent TTPs by using, respectively, verifiable encryption (which however is computationally inefficient) and designated convertible signatures (requiring an additional interactive protocol, which is not efficient).

In this paper, we propose a new optimistic fair exchange protocol based on the Guillou-Quisquater signature scheme that uses an offline TTP producing the same digital signatures as the client and the provider would produce in a faultless case. Our protocol is more efficient than the previously cited protocols. A similar protocol [3] based on the GPS signature scheme [4] has been recently proposed and we show here that another efficient instance of the proposed protocol can be derived from the Guillou-Quisquater signature scheme. The particularity of the present scheme is that it is identity-based. This implies a more efficient protocol, in the sense that no certificates on the public keys are needed, hence reducing the communication and computational cost required by the parties. The price to pay is, however, a stronger trust requirement. In this framework, the TTP is more powerful than in certificate-based or self-certified schemes, but we propose a way to limit this power by introducing several TTP's in the system. We assume that the communication channel between the provider and the client is unreliable (the transmitted data may be lost or modified), and the communication channels between the provider and the TTP, and also between the client and the TTP are resilient (the transmitted data is delivered after a finite, but unknown amount of time; the data may be delayed, but will eventually arrive).

2 A fair exchange protocol

The proposed protocol is based on the Guillou-Quisquater signature scheme that may be described as follows. The TTP chooses two large random primes p and q and a relatively large prime v, co-prime with $p-1$ and $q-1$. It also selects a collision-resistant hash function h whose output is $< v$ and almost of the same size. Then, it makes h, v and $n = pq$ public and keeps p and q secret. The secret key x of a user is computed as $x = I^{-v^{-1}} \pmod n$, where I is a string describing the user's identity and serves as his public key and v^{-1} is computed modulo $\lambda(n)$. To generate a signature on a message m, the signer chooses a random integer $r \in Z_n$ and computes $t = h(r^v \bmod n, m)$ and $T = r \cdot x^t \bmod n$. The signature is the pair (t, T) and is accepted as valid if $t = h(T^v \cdot I^t \bmod n, m)$. In our scheme, in addition to the parameters above, the TTP selects a very small public prime $e \neq v$ that is also co-prime with $p-1$ and $q-1$.

We will use the following notations in our protocol: $item$ is the electronic item to be transmitted to the client and $descr$ is a string containing the client's request, the description of the requested item and some other information allowing the provider, the TTP and any other external party to recognize the item. C, P, TTP identify, respectively, the client, the provider and the trusted third party. $h(X)$ is the output of a one-way hash function h applied to the message X. $\mathrm{S}_P(X)$ is a "classical" provider's digital signature (and not recoverable) of the message X. $\mathrm{E}_k(X)$ is a symmetric encryption of the message X with the session key k and $\mathcal{E}_{TTP}(X)$ is an asymmetric encryption of the message X with the TTP's public key. f_{xxx} are flags indicating the purpose of a message sent. l is a label identifying the protocol run, together with the identities P and C.

Main protocol

When a client and a provider agree to exchange a provider's item against the client's signature, they follow this protocol.

1. The provider selects a random value r_P and computes $t_P = r_P^{ev} \bmod n$ and $T_P = r_P^e \cdot x_P^{h(t_P, m_P)}$ where $m_P = (f_{msg}, P, C, l, descr)$. He also selects a random session key k and forms $\mathrm{E}_k(l, item)$ and $\mathcal{E}_{TTP}(k)$. The pair (t_P, T_P), being the provider's committed signature, is sent to the client together with f_{com_1}, P, C, l, $descr$, $\mathrm{E}_k(l, item)$, $\mathcal{E}_{TTP}(k)$ and the provider's classical signature on those information.

2. The client checks that the provider's committed signature may be opened, if necessary, by the TTP to provide the final signature and that the provider's signature on the ciphered information is valid. The client forms m_P on his own side and checks whether the provider's classical signature is valid and whether $t_P \equiv T_P^v \cdot I_P^{h(t_P, m_P)} \bmod n$. If so, the client chooses a random r_C and computes $t_C = r_C^{ev} \bmod n$ and $T_C = r_C^e \cdot x_C^{h(t_C, m_C)}$ where $m_C = (f_{ack}, C, P, l, descr)$. The client's committed signature, composed by the pair (t_C, T_C), is sent to the provider together with f_{com_2}, C, P and l.

3. The provider forms m_C and checks whether $t_C \equiv T_C^v \cdot I_C^{h(t_C, m_C)} \bmod n$. If so, the provider computes $t'_P = r_P^v \bmod n$ and sends to the client the item (or just the session key, in order to decrease the amount of communications), t'_P, f_{msg}, P, C and l. The pair (t'_P, T_P) being the provider's final signature.

4. The client verifies that $t'^e_P \equiv T_P^v \cdot I_P^{h(t'^e_P \bmod n, m_P)} \bmod n$. If so, after having checked the validity of the received item, the client computes $t'_C = r_C^v \bmod n$ and sends to the provider f_{ack}, C, P, l and t'_C. The pair (t'_C, T_C) being the final signature.

5. The provider verifies that $t'^e_C \equiv T_C^v \cdot I_C^{h(t'^e_C \bmod n, m_C)} \bmod n$. If so, the provider accepts the signature, since it will also be accepted by any external party.

Provider's recovery protocol

If the client does not send his final signature or if the one transmitted is not valid, the provider runs the following protocol with the TTP in order to recover the client's final signature.

1. The provider sends to the TTP the flag f_{rec_P}, P, C, l, $descr$, $item$, t_P, his final signature (t'_P, T_P) and the pair (t_C, T_C).

2. If the protocol has already been recovered or aborted, the TTP stops the recovery protocol. Otherwise, the TTP makes sure that the item corresponds actually to its description and if so, it forms m_C and m_P and verifies the validity of (t_C, T_C) and (t'_P, T_P). If all the checks are successful (1) the TTP sends $t'_C = t_C^{e^{-1}} \bmod n$ to the provider together with f_{ack}, C, P and l, (2) and the TTP sends also f_{msg}, P, C, l, $item$ and t'_P to the client. Otherwise, it sends an abort token to both parties.

Client's recovery protocol

If the provider does not send the item and his final signature to the client, or if the transmitted information is not valid, the client runs the following protocol with the TTP.

1. The client sends f_{rec_C}, C, P, l, $descr$, $\mathrm{E}_k(l, item)$, $\mathcal{E}_{TTP}(k)$, $\mathrm{S}_P(f_{com_1}, P, C, l, \mathrm{E}_k(l, item), \mathcal{E}_{TTP}(k))$, the pair (t_P, T_P) (the provider's committed signature), t_C and his final signature (t'_C, T_C) to the TTP.

2. If the protocol has already been recovered or aborted, the TTP stops the recovery protocol. Otherwise, the TTP

first makes sure that the ciphered label and item are coherent, that the received item (obtained after deciphering) corresponds actually to $descr$ and that the provider's signature $sigP$ is valid, if so it forms m_C and m_P and verifies the validity of (t_P, T_P) and (t_C, T_C). If the signatures are invalid the TTP stops the recovery protocol (but does not send an abort token). If the other checks are not successful the TTP sends an abort token to the provider and to the client, as in the abort protocol. Otherwise, if the checks are successful, the TTP sends f_{msg}, P, C, l, $item$ and $t'_P = {t_P}^{e^{-1}} \bmod n$ to the client; and f_{ack}, C, P, l together with the pair $(t'_C = {t_C}^{e^{-1}}, T_C)$ to the provider.

Abort protocol

If the client does not send his committed signature or if the one transmitted is not valid, the provider runs the following protocol with the TTP, in order to abort the protocol.

1. The provider sends an abort request, composed of f_{ab1}, C, P, l and $\mathrm{S}_P(f_{ab1}, C, P, l)$, to the TTP.

2. If the protocol was not already recovered or aborted, the TTP sends an abort confirmation to the provider and the client. The TTP sends f_{ab2}, P, C, l and $\mathrm{S}_{TTP}(f_{ab2}, P, C, l)$ to both the provider and the client.

Fairness

If the client stops the main protocol after receiving the first message, either the client can run a recovery protocol with the TTP during which the client and the provider receive their expected information, or the provider can run an abort protocol with the TTP, in which case neither the client nor the provider receive their expected information. In both cases the protocol remains fair. If the provider stops the main protocol after receiving the client's committed signature, then the only possible way for him to get the client's final signature is to run a recovery protocol with the TTP. In this case, both the client and the provider receive their expected information and so the protocol remains fair. If the client stops the main protocol after receiving the item, the provider can initiate a recovery protocol with the TTP that sends him the client's final signature and the item to the client. The protocol is remaining fair, due to the resilient channels between the TTP and respectively the provider and the client.

3 Security analysis

The security of our protocol may be discussed around two questions: (1) Is it possible to forge committed signatures linked to a given user? (2) Is it possible to forge final signatures linked to a given user directly (i.e., without having the related committed signature) or by converting a committed signature to a final one without knowing r or e^{-1}?

Theorem 1: If the Guillou-Quisquater signature scheme is secure then it is infeasible to forge a committed signature without knowing the user's secret key. Proof: A committed signature in our scheme is exactly the same as a Guillou-Quisquater signature in the sense that it is verified the same way. So if it is possible to forge a committed signature on a message m without knowing the secret key, then it would be possible to create a Guillou-Quisquater signature on the same message by the same means.

Theorem 2: If the Guillou-Quisquater signature scheme is secure then it is infeasible to forge a final signature without knowing the user's secret key or from the related committed signature without knowing r or e^{-1}. Proof: If it is possible to create a final signature (t', T) on a message m just from the user's public key and known signatures (but without having the corresponding committed signature), then it would be possible to convert it to $(t = {t'}^e \bmod n, T)$, as a committed signature that is actually a Guillou-Quisquater signature on the same message.

On the other hand, if we have a committed signature (t, T) on a message m, in order to compute t', one should either know r (to do as the real client does) or e^{-1} (to do like the TTP). However, suppose that it would be possible for a cheater to use (t, T) to create a correct final signature of the form $(\hat{t'}, \hat{T})$. This signature may be such that $\hat{T} \neq T$ (that implies that $\hat{t'} \neq t'$) or $\hat{T} = T$ but $\hat{t'} \neq t'$, where (t', T) is the real final signature derived from (t, T). We prove that in either case, establishing a final signature from t and T is equivalent to forging a Guillou-Quisquater signature. In fact, if one generates a valid pair $(\hat{t'}, \hat{T})$ from (t, T), then it is possible to form another committed signature $(\hat{t} = {\hat{t'}}^e, \hat{T})$ for the same message. However, both committed signatures are valid Guillou-Quisquater signatures. This means that we have an algorithm that given a valid Guillou-Quisquater signature on a message m, outputs another Guillou-Quisquater signature on the same message: $(t, T) : T^v \cdot I^{h(t,m)} = t \Rightarrow (\hat{t}, \hat{T}) : \hat{T}^v \cdot I^{h(\hat{t},m)} = \hat{t}$. But, this is actually equivalent to forging a Guillou-Quisquater signature on a new message m'. Indeed, since h is, by assumption, a collision-resistant one-way hash function, it may be considered as a random function. This means that $h(\hat{t}, m)$ is random and its value is not predictable before computing $\hat{t}$. So, if the algorithm can create a Guillou-Quisquater signature $(\hat{t}, \hat{T})$ on a message m, a modification of that algorithm would create a signature on another message m'. As noted in the proof of theorem 1, a committed signature is verified in the same way as a regular Guillou-Quisquater signature. So, if a client deviates from the protocol and computes his committed signature as $t_C = {r_C}^v \bmod n$ and $T_C = r_C \cdot {x_C}^{h(t_C, m_C)} \bmod n$, the protocol still remains secure, because a recovery by the TTP (requested by a provider) on this signature yields straight-

forwardly a correct final signature.

4 Working with several TTP's

As is the case in all identity-based schemes, the user secret keys are calculated by the TTP. This means that the TTP can potentially do anything with these keys. So, a basic requirement in our scheme is that the authority should be fully trusted by all the users. This assumption may be acceptable for local applications with a reduced scale. For many other applications covering users around the world, however, trusting a single central authority is too strong and surely not realistic. For this purpose, we may conceive a system made up by several TTPs that each has its own setting parameters; i.e., p, q and consequently n (the public values v, e and the hash function h may remain common to all TTPs). Each user is associated to a given TTP that is responsible of generating its pair of keys and is the only entity (other than the user itself) that knows it. In this case, when the users participate in a fair-exchange protocol, they have first to ensure that the TTP of the other party exists before the start of the main protocol. When the users follow the predetermined protocol and if there is no problem due to the network, then the protocol is exactly the same as in the case with a single TTP. If the provider does not send the item and his final signature to the client, or if the transmitted information is not valid, (1) the client sends the received ciphered information, the description of the item, the provider's signature on them, the pair (t_P, T_P) (the provider's committed signature) and his final signature (t'_C, T_C) to the provider's TTP, (2) the provider's TTP verifies whether the protocol has previously been recovered by the client. If not so, it sends a request to the client's TTP to ask if the protocol has been recovered or aborted by the provider. Upon receiving the reply of the client's TTP, the provider's TTP first makes sure that the item is ciphered with a coherent label, that the received item (obtained after deciphering) corresponds actually to the received description and that the provider's signature $sigP$ is valid, if so it forms m_C and m_P and verifies the validity of (t_P, T_P) and (t'_C, T_C). If the signatures are invalid the TTP stops the recovery protocol (but does not send an abort token). If the other checks are not successful the TTP sends an abort token to the provider and to the client, as in the abort protocol. Otherwise, if all the checks are successful, the TTP sends $t'_P = {t_P}^{e^{-1}} \bmod n$ and the item to the client and (t'_C, T_C) to the provider. A similar mechanism is realized if the client does not send his final signature or if the one transmitted is not valid.

The interesting question that arises from the discussion above is, "if the TTP knows or can compute user secret keys, what is the advantage of our protocol". Indeed, one can imagine a simple protocol using classical signatures and where the TTP, if requested, knowing the secret keys, can produce the final signatures. Such a mechanism is always conceivable and functions with any existing digital signature scheme. However, this way of conceiving the fair exchange implies that the TTP should record all the user secret keys or at least it has to recompute them when necessary, as well as the complete final signatures in case of problem. This is what we wanted to avoid for free. In our protocol, the TTP has only to save a single secret key e^{-1}, with which it can produce transparently the final digital signatures requiring only one modular exponentiation.

5 Conclusion

We have considered a new fair exchange protocol allowing the exchange of an item against a signature while assuring fairness. Our protocol uses an offline and invisible trusted third party. During the protocol, committed signatures are issued, giving sufficient assurance to the recipients about the TTP's ability of recovering the final signatures from the committed ones. As the TTP is invisible, it is able to produce, the same final signature as the one transmitted in a faultless case, rather than an affidavit or an official certificate. An interesting feature of our method is the low communication and computational charges required by the parties during the protocol. Moreover, as it is difficult to determine whether the TTP was requested during the protocol because of a dishonest party or because of a network problem, an invisible TTP may be particularly relevant, for example, in an electronic commerce environment.

References

[1] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. In *Proceedings of Eurocrypt'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 591–606.

[2] C. Boyd and E. Foo. Off-line fair payment protocols using convertible signatures. In *Proceedings of Asiacrypt'98*, volume 1514 of *Lecture Notes in Computer Science*, pages 271–285.

[3] O. Markowitch and S. Saeednia. Optimistic fair-exchange with transparent signature recovery. In *Proceedings of Financial Cryptography 2001*, volume 2339 of *Lecture Notes in Computer Science*, pages 339–350.

[4] G. Poupard and J. Stern. Security analysis of a practical "on the fly" authentication and signature generation. In *Proceedings of Eurocrypt'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 422–436.