2010

An abuse-free fair contract-signing protocol based on the RSA signature

Guilin Wang
University of Birmingham, guilin@uow.edu.au

Follow this and additional works at: https://ro.uow.edu.au/infopapers

Part of the Physical Sciences and Mathematics Commons

Recommended Citation

Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au
An abuse-free fair contract-signing protocol based on the RSA signature

Abstract
A fair contract-signing protocol allows two potentially mistrusted parities to exchange their commitments (i.e., digital signatures) to an agreed contract over the Internet in a fair way, so that either each of them obtains the other’s signature, or neither party does. Based on the RSA signature scheme, a new digital contract-signing protocol is proposed in this paper. Like the existing RSA-based solutions for the same problem, our protocol is not only fair, but also optimistic, since the trusted third party is involved only in the situations where one party is cheating or the communication channel is interrupted. Furthermore, the proposed protocol satisfies a new property- abuse-freeness. That is, if the protocol is executed unsuccessfully, none of the two parties can show the validity of intermediate results to others. Technical details are provided to analyze the security and performance of the proposed protocol. In summary, we present the first abuse-free fair contract-signing protocol based on the RSA signature, and show that it is both secure and efficient.

Disciplines
Physical Sciences and Mathematics

Publication Details
An Abuse-Free Fair Contract-Signing Protocol Based on the RSA Signature

Guilin Wang

Abstract—A fair contract-signing protocol allows two potentially mistrusted parties to exchange their commitments (i.e., digital signatures) to an agreed contract over the Internet in a fair way, so that either of each of them obtains the other’s signature, or neither party does. Based on the RSA signature scheme, a new digital contract-signing protocol is proposed in this paper. Like the existing RSA-based solutions for the same problem, our protocol is not only fair, but also optimistic, since the trusted third party is involved only in the situations where one party is cheating or the communication channel is interrupted. Furthermore, the proposed protocol satisfies a new property—abuse-freeness. That is, if the protocol is executed unsuccessfully, none of the two parties can show the validity of intermediate results to others. Technical details are provided to analyze the security and performance of the proposed protocol. In summary, we present the first abuse-free fair contract-signing protocol based on the RSA signature, and show that it is both secure and efficient.

Index Terms—Contract signing, cryptographic protocols, digital signatures, e-commerce, fair-exchange, RSA, security.

I. INTRODUCTION

Contract signing plays a very important role in any business transaction, in particular in situations where the involved parties do not trust each other to some extent already. In the paper-based scenario, contract signing is truly simple due to the existence of “simultaneity.” That is, both parties generally sign two hard copies of the same contract at the same place and at the same time. After that, each party keeps one copy as a legal document that shows both of them have committed to the contract. If one party does not abide by the contract, the other party could provide the signed contract to a judge in court.

As electronic commerce is becoming more and more important and popular in the world, it is desirable to have a mechanism that allows two parties to sign a digital contract via the Internet. However, the problem of contract signing becomes difficult in this setting, since there is no simultaneity any more in the scenario of computer networks. In other words, the simultaneity has to be mimicked in order to design a digital contract-signing protocol. This requirement is essentially captured by the concept of fairness: At the end of the protocol, either both parties have valid signatures for a contract or neither does, even if one of them tries to cheat or the communication channel is out of order. In fact, Even and Yacobi [22] proved that it is impossible to achieve fairness in a deterministic two-party contract-signing protocol. The intuitive reason could be explained as follows. The purpose of such a protocol is to go from the initial fair state, in which no party has what he/she expects, to the desired fair state in which both obtain what they want. However, information is exchanged in computer networks nonsimultaneously, so at least an unfair state must be passed through.

Related Work: From the view point of technique, the problem of digital contract signing belongs to a wider topic: fair exchange, i.e., how to enable two (or multiple) potentially mistrusted parities exchanging digital items over public computer networks like the Internet in a fair way, so that each party gets the other’s item, or neither party does. Actually, fair exchange includes the following different but related issues: contract-signing protocols [2], [4], [6], [7], [12], [17], [22], [26], [39], certified e-mail systems [1], [5], [32], [35], [49], nonrepudiation protocols [31], [36], [46], [48], and e-payment schemes in electronic commerce [15], [40]. For more references and discussions on the relationships between those conceptions, please refer to [3], [36], and [46]. In this paper, we mainly focus on the problem of digital contract signing between two parties. Since a party’s commitment to a digital contract is usually defined as his/her digital signature on the contract, digital contract signing is essentially implied by fair exchange of digital signatures between two potentially mistrusted parities.

There is a rich history of contract signing (i.e., fair exchange of digital signatures) because this is a fundamental problem in electronic transactions. According to the involvement degree of a trusted third party (TTP), contract-signing protocols can be divided into three types: 1) gradual exchanges without any TTP; 2) protocols with an on-line TTP; and 3) protocols with an off-line TTP. Early efforts [17], [21], [29] mainly focused on the first type of protocols to meet computational fairness: Both parties exchange their commitments/secrets “bit-by-bit.” If one party stops prematurely, both parties have about the same fraction of the peer’s secret, which means that they can complete the contract off-line by investing about the same amount of computing work, e.g., exclusively searching the remaining bits of the secrets. The major advantage of this approach is that no TTP is involved. However, this approach is unrealistic for most real-world applications due to the following reasons. First of all, it is assumed that the two parties have equivalent or related computation resources. Otherwise, such a protocol is favorable to the party with stronger computing power, who may conditionally force the other party to commit the contract by its
own interest. At the same time, such protocols are inefficient because the costs of computation and communication are extensive. In addition, as pointed out in [12], this approach has the unsatisfactory property of uncertain termination. For example, suppose two parties are signing a house-sale contract. If the protocol stops prematurely on the side of the buyer, the seller will never be sure whether the buyer is continuing with the protocol, or has terminated—and perhaps even has engaged in another house-sale contract-signing protocol with another seller. The buyer may be in a similar situation if the protocol terminated on the side of the seller.

In the second type of fair exchange protocols [12], [18], [48], an on-line TTP is always involved in every exchange. In this scenario, a TTP is essentially a mediator: a) Each party first sends his/her item to the TTP; b) then, the TTP checks the validity of those items; c) if all expected items are correctly received, the TTP finally forwards each item to the party who needs it. Generally speaking, contract-signing protocols with an on-line TTP could be designed more easily since the TTP facilitates the execution of each exchange, but may be still expensive and inefficient because the TTP needs to be paid and must be part of every execution (though maybe not involved in each step). In practice, the on-line TTP is prone to become a bottleneck in the whole system, especially in the situation where many users rely on a single TTP.

Compared with the schemes belonging to the previous two types, contract-signing protocols with off-line TTP [2], [3], [4], [6], [40] are more appealing and practical for most applications because those protocols are optimistic in the sense that the TTP is not invoked in the execution of exchange unless one of the two parties misbehaves or the communication channel is out of order. Bao et al. [6] and Ateniese [4] constructed fair exchange protocols of digital signatures from verifiably encrypted signatures, while Asokan et al. [2], [3] proposed such protocols by using verifiable escrows. The basic ideas behind those two cryptographic primitives are similar, as explained below. To get the digital signature from the other party, Bob, a party, Alice, first encrypts her signature under the TTP’s public encryption key, and proves to Bob that the ciphertext indeed corresponds to her signature, interactively or noninteractively. Then, Bob sends his digital signature (or some digital item) to Alice. After receiving the expected item from Bob, Alice reveals her signature to Bob. The point is that if Alice refuses to do so after getting Bob’s item, the TTP can decrypt Alice’s encrypted signature and sends the result to Bob. The difference between those two kinds of schemes is that in the verifiable escrow-based schemes, Alice, the creator of the encryption, has the ability to control the conditions under which the encryption could be decrypted by the TTP. Though their techniques can be applied to a variety of signature schemes, the overheads of computation and communication are usually expensive. In particular, the schemes in [2], [3], and [6] are inefficient, since expensive cut-and-choose techniques [23] are used to prove the correctness of the encrypted signature. In addition, it is noticed in [8] that the Schnorr and ElGamal signature-based fair-exchange schemes in [4] should be improved to avoid a security flaw.

In [39], Micali constructed several simple fair exchange schemes based on any secure signature and encryption algorithms. However, Bao et al. [7] pointed out that his contract-signing protocol is actually unfair because there is an intrinsic flaw in the dispute resolution protocol, i.e., the policy exploited by the TTP to settle potential disputes between the two parties involved in a contract signing.

Based on an RSA multisignature scheme, Park et al. [40] proposed a novel fair exchange protocol with an off-line trusted party. Their protocol was fair and optimistic but insecure, since Dodis and Reyzin [20] broke their protocol by pointing out that an honest-but-curious TTP can easily derive a user’s private key after the end of his/her registration. Moreover, as an improvement of Park et al.’s scheme, Dodis et al. [20] even constructed a provably secure fair exchange protocol from the noninteractive two-signature one of Boldyreva [13]. Their scheme works in gap Diffie–Hellman (GDH) groups (refer to [45] for an explanation). The pairing-based cryptosystems [13], [14] are typical examples constructed from GDH groups. However, note that in such cryptosystems, the computation of the pairing is still time-consuming, although several papers have investigated speeding up the pairing computation [9], [25].

Furthermore, we remark that, in essence, Dodis et al.’s scheme is not an improvement of Park et al.’s scheme, since the security of their scheme is based on the GDH problem instead of the RSA probem or factoring problem [42]. As the RSA cryptosystem [42] is now the de facto industrial standard and is widely used in many applications, it is highly desirable to construct fair exchange protocols based on RSA. Actually, as we mentioned before, several such schemes have been proposed: Asokan et al.’s scheme [2], [3] from verifiable escrow, Ateniese’s scheme [4] from verifiably encrypted signature, and Park et al.’s scheme [40] from multisignature. However, all those schemes are not abuse-free [26]. That is, a party can get verifiable intermediate results when the signature exchange protocol is executed unsuccessfully. Consequently, this party may obtain some benefits by showing such universally verifiable intermediate results to a third party. For example, Bob is looking for a job and he has received two offers from competing companies A and C. Bob prefers to join company C though the offered salary is not satisfactory. In contrast, company A promises a higher salary but he does not really like to join it due to some personal reason, such as weather, culture, or something else. In this scenario, Bob may first pretend to sign an employment contract with company A. Then, he terminates the execution of the contract-signing protocol after he obtained the intermediate results generated by company A. By showing such universally verifiable proofs to company C, Bob may get a higher salary from company C. There exists the same problem in other similar situations.

Therefore, running contract protocols without the property of abuse-freeness is a risk for a honest party, as a possible dishonest party maybe does not really want to sign the contract with her, but only use her willingness to sign to get leverage for another contract. Consequently, this is an important security requirement for contract-signing protocols, especially in the situations where partial commitments to a contract may be beneficial to a dishonest party or an outsider. However, except the discrete logarithm-based scheme of Garay et al. [26], all other optimistic contract-signing protocols [2], [3], [4], [6], [7], [39], [40] are not abuse-free.
Our Work: Motivated by the above example that shows the importance of abuse-freeness, and the question of how to improve Park et al.’s scheme in a secure way, this paper proposes a new contract-signing protocol for two mutually distrusted parties. Our protocol is based on an RSA multisignature, which is formally proved to be secure by Bellare and Sandhu [11]. Like the schemes in [2], [4], and [40], our protocol is fair and optimistic. Furthermore, different from the above existing schemes, our protocol is abuse-free. The reason is that we integrate an interactive zero-knowledge protocol, proposed for confirming RSA undeniable signatures by Gennaro et al. [27], into our scheme to prove the validity of the intermediate results. Moreover, we exploit trapdoor commitment schemes to enhance this zero-knowledge protocol so that the abuse-freeness property can be fully achieved. Technical analysis and discussion are provided in detail to show that our scheme is secure and efficient.

More specifically, the new protocol satisfies the following desirable properties.

1) **Fairness**: Our protocol guarantees the two parties involved to obtain or not obtain the other’s signature simultaneously. This property implies that even a dishonest party who tries to cheat cannot get an advantage over the other party.

2) **Optimism**: The TTP is involved only in the situation where one party is cheating or the communication channel is interrupted. So it could be expected that the TTP is only involved in settling disputes between users rarely, due to the fact that fairness is always satisfied, i.e., cheating is not beneficial to the cheater.

3) **Abuse-Freeness**: If the whole protocol is not finished successfully, any of the two parties cannot show the validity of the intermediate results generated by the other to an outsider, either during or after the procedure where those intermediate results are produced. As we mentioned before, the unique known abuse-free contract-signing protocol [26] is based on the discrete logarithm problem, instead of the RSA cryptosystem.

4) **Provable Security**: Under the standard assumption that the RSA problem is intractable [11], [42], the protocol is provably secure in the random hash function model [10], where a hash function is treated as if it were a “black box” containing a random function.

5) **Timely Termination**: The execution of a protocol instance will be terminated in a predetermined time. This property is implemented by adding a reasonable deadline t in a contract, as suggested by Micali in [39]. If one party does not send his/her signature to the other party after the deadline t, both of them are free of liability to their partial commitments to the contract and do not need to wait any more.

6) **Compatibility**: In our protocol, each party’s commitment to a contract is a standard digital signature. This means that to use the protocol in existing systems, there is no need to modify the signature scheme or message format at all. Thus, it will be very convenient to integrate the contract-signing protocol into existing software for electronic transactions.

7) **TTP’s Statelessness**: To settle potential disputes between users, the TTP is not required to maintain a database to searching or remembering the state information for each protocol instance, so the overhead on the side of the TTP is reduced greatly, compared with the previous schemes in [2], [3], and [26].

8) **High Performance**: In a typical implementation, the protocol execution in a normal case requires only interaction of several rounds between two parties, transmission of about one thousand bytes of data, and computation of a few modular exponentiations by each party.

The rest of the paper is organized as follows. Section II reviews Park et al.’s scheme and its security. We then introduce trapdoor commitment schemes in Section III, as they are crucial for fully archiving abuse-freeness. Section IV presents our new contact signing protocol based on the RSA signature. After that, we analyze its security and efficiency in Sections V and VI, respectively. Finally, Section VII gives the conclusion.

II. PARK ET AL.’S SCHEME AND ITS SECURITY

In this section, we briefly overview Park et al.’s scheme and the attack on it identified by Dodis and Reyzin. For more detail, please refer to the original papers [20], [40].

In Park et al.’s scheme, Alice sets an RSA modulus n = pq, where p and q are two k-bit safe primes, and picks her random public key e ∈ Z_p^∗, and calculates her private key d = e^(-1) mod φ(n), where φ(n) = (p - 1)(q - 1) is Euler’s totient function. Then, she registers her public key with a certification authority (CA) to get her certificate C_A. After that, Alice randomly splits d into d_1 and d_2 so that d = d_1 + d_2 mod φ(n), where d_1 ∈ Z_p^∗. To get a voucher V_A from a TTP, Alice is required to send (C_A, e_1, d_2) to the TTP, where e_1 = d_1^(-1) mod φ(n). The voucher V_A is the TTP’s signature that implicitly shows two facts: 1) e_1 can be used to verify a partial signature generated by using secret key d_1, and 2) the TTP knows a secret d_2 that matches with RSA key pairs (d_1, e_1) and (d, e).

When Alice and Bob want to exchange their signatures on a message m, Alice first computes σ_1 = h(m)^{d_1} mod n, and sends ⟨C_A, V_A, σ_1⟩ to Bob, where h(·) is a secure hash function. Upon receiving ⟨C_A, V_A, σ_1⟩, Bob checks the validity of C_A and V_A, and whether h(m) ≡ σ_1^e mod n. If all those verifications go through, Bob returns his signature σ_B to Alice, since he is convinced that the expected σ_2 = h(m)^{d_2} mod n can be revealed by Bob or the TTP. After receiving valid σ_B, Alice reveals σ_2 = h(m)^{d_2} mod n to Bob. Finally, Bob obtains Alice’s signature σ_A for message m by setting σ_A = σ_1 σ_2 mod n, since we have

\[ h(m) ≡ \sigma_A = h(m)^{d_1 + d_2} = \sigma_1^{d_1} h(m)^{d_2} \mod n. \]

The security problem in Park et al.’s scheme is that an honest-but-curious TTP can easily derive Alice’s private key d. The reason is that with the knowledge of ⟨n, e, e_1, d_2⟩, the TTP knows that the integer \( e - (1 - e d_2)e_1 \) is a nonzero multiple of
It is well known that knowing such a multiple of $\phi(n)$, Alice’s RSA modulus $n$ can be easily factored. Consequently, the TTP can get Alice’s private key $d$ by the extended Euclidean algorithm.

The point is that we do not want the TTP to have the ability of making a user’s signature independently, though the TTP is a (partially) trusted party. The main reason is that as the pivotal secret of any cryptosystem, the private key should not be revealed to any party, including a partially trusted party. In addition, if there is a completely trusted TTP, the problem of fair exchange can be solved trivially as follows. First, each party gives his/her private key to the TTP before exchanging items so that the TTP can generate signatures on behalf of any party if necessary. Then, the TTP issues a voucher for each registered party to show that it knows this party’s private key. When Alice and Bob want to exchange their signatures on a message $m$, they first exchange their vouchers issued by the TTP. By doing so correctly, it is proved that both of them have registered with the TTP. After that, their signatures can be delivered directly to the other side. If one party, say Alice, does not receive Bob’s signature on $m$, she applies the TTP’s help by providing her signature and message $m$. After checking the correctness of this information, the TTP will generate and send Bob’s signature on $m$ to Alice by using Bob’s private key.

III. TRAPDOOR COMMITMENT SCHEMES

As using standard zero-knowledge is not enough to guarantee the abuse-freeness in our protocol, we need another cryptographic primitive, called trapdoor commitment schemes. So we now introduce this important concept and review two popular and very efficient schemes, based on RSA and discrete logarithm problems, respectively.

As a two-phase protocol running between a sender and a receiver, a commitment scheme [16], [41] allows the sender to first hide a value by computing a commitment, and then reveals the hidden value together with some related information to open the commitment so that the receiver can check whether the commitment is decommitted correctly. Informally, a secure commitment scheme should satisfy the binding property and the hiding property. The former means that given a commitment, the receiver is unable to know which value is committed, while the latter requires that once a commitment has been made, the sender cannot change his mind to cheat the receiver by revealing a different value, which is not the value committed initially.

In a trapdoor commitment (TC) scheme [24], [28], [37], there is one trapdoor that would allow the owner of this trapdoor to open a commitment in different ways. Due to this amazing additional property, a valid answer to a commitment can only be accepted by the owner of the trapdoor, usually the commitment receiver. The reason is that once getting such a valid answer, an outsider cannot distinguish whether this answer is revealed by the sender or forged by the receiver using the trapdoor. Actually, this is why trapdoor commitment schemes can help us to achieve the abuse-freeness property in the contract-signing scenario.

Formally, a trapdoor commitment scheme $\mathcal{TC}$ consists of four algorithms, i.e., $\mathcal{TC} = \langle \mathcal{TC}_{\text{gen}}, \mathcal{TC}_{\text{com}}, \mathcal{TC}_{\text{ver}}, \mathcal{TC}_{\text{sim}} \rangle$. The receiver, say Bob, runs the key generation algorithm $\mathcal{TC}_{\text{gen}}$ to get a commitment public key $pk$ and the corresponding trapdoor $td$. Given a value $r$ and the commitment public key $pk$, commitment algorithm $\mathcal{TC}_{\text{com}}$ outputs a pair $(\text{com}, \text{dec})$, where com is the commitment to value $r$ and $\text{dec}$ is the related information used to decommit $\text{com}$. A commitment verification algorithm $\mathcal{TC}_{\text{ver}}$ is used to check whether an answer $(r, \text{dec})$ is valid to a given commitment $\text{com}$ w.r.t. public key $pk$. Finally, a simulation algorithm allows the receiver Bob, using the trapdoor $td$, to simulate a new answer $(r’, \text{dec‘})$ for a commitment $\text{com}$ when one answer $(r, \text{dec})$ for $\text{com}$ is given.

Note that theoretically any secure digital signature implies a secure trapdoor commitment scheme. This result can be easily obtained from [24], [37], and [43], as it is shown in [43] that the existence of secure signatures is equivalent to the existence of one-way functions, while [24] and [37] report how to construct trapdoor commitment schemes from any one-way functions. This means that for any kind of public key held by the sender, say Alice, to achieve abuse-freeness in our contract-signing protocol. In the following, we just show how efficient and secure trapdoor commitment schemes can be constructed from RSA and discrete logarithm related problems, as the corresponding signature schemes are very popular.

A. Strong RSA-Based Trapdoor Commitment Scheme

The following RSA-based efficient trapdoor commitment scheme is proposed by Gennaro in [28].

1) $\mathcal{TC}_{\text{gen}}$: The receiver Bob first generates two large primes $p_B$ and $q_B$, sets an RSA modulus $n_B = p_Bq_B$, selects a random number $s \in \mathbb{Z}_{n_B}^*$, picks a 160-bit prime number $u$ such that $\text{GCD}(u, \phi(n_B)) = 1$, and selects a collision-resistant hash function $h_2 : \{0, 1\}^\ast \rightarrow \{0, 1\}^{130}$. Then, $\mathcal{TC}_{\text{gen}}$ outputs the commitment public key $pk = (n_B, s, u, h_2)$ and the trapdoor $td = \rho_B$, where $\rho_B \in \mathbb{Z}_{n_B}^*$ is the $u$th root of $s$, i.e., $\rho_B^u = s \mod n_B$. (Alternatively, the factors of $n_B$ can be used as trapdoor.)

2) $\mathcal{TC}_{\text{com}}$: To commit to a string $r$ with arbitrary length, the sender sends the receiver $\text{com} \leftarrow s^{h_2(r)}u^t \mod n_B$, where $t \in \mathbb{Z}_{n_B}^*$ is a randomness, and stores $\text{dec} = t$.

3) $\mathcal{TC}_{\text{ver}}$: To decommit $\text{com}$, the sender reveals $(r, t)$, so that the receiver can check if $\text{com} \equiv s^{h_2(r)}u^t \mod n_B$.

4) $\mathcal{TC}_{\text{sim}}$: Given an answer $(r, t)$ to a commitment $\text{com} \equiv r$, by using the trapdoor $\rho_B$, the receiver Bob can decommit $\text{com}$ w.r.t. any string $r’$ by revealing $(r’, t’)$, where $t’ = \rho_B^{h_2(r’)-h_2(r)} \cdot t \mod n_B$. It is easy to see that $(r’, t’)$ is also a valid answer to the commitment $\text{com}$.

In [28], the above trapdoor commitment scheme is formally proved to be secure under the strong RSA assumption, which says that given a random element $s \in \mathbb{Z}_{n_B}^*$, it is infeasible to find a pair $(\rho, u \neq 1)$ such that $\rho^u = s \mod n_B$.

Note that in the above trapdoor commitment scheme the parameters $s$ and $u$ can be shared by multiple receivers. For example, we can let $s = 2$ and $u$ a fixed 160-bit prime number for all receivers who employ RSA signatures. In this way, a receiver Bob’s standard RSA public key implicitly defines a trapdoor commitment scheme. Therefore, a sender Alice who only knows Bob’s RSA public key can run the above trapdoor commitment scheme without enquiring the values of $s$ and $u$, and
the receiver Bob is also not required to run an extra commitment key generation algorithm, though Bob may need to extract the trapdoor $p_B$ when necessary. This feature simplifies our new contract-signing protocol.

B. DL-Based Trapdoor Commitment Scheme

As pointed out in [24], Pedersen’s commitment scheme [41] can be easily extended into a trapdoor commitment scheme, whose security relies on the discrete logarithm (DL) problem.

1) $TC_{\text{Gen}}$: The receiver Bob first generates a large prime $p_B$, picks a generator $g$ for the subgroup $G \subseteq \mathbb{Z}_p^*$ of prime order $q_B$, where $q_B|p_B-1$ and $|G| = 160$, selects a random number $x_B \in_R \mathbb{Z}_{q_B}$, sets $y_B = g^{x_B} \mod p_B$, and chooses a collision-resistant hash function $h_2 : \{0,1\}^* \rightarrow \{0,1\}^{163}$. Then, $TC_{\text{Gen}}$ outputs the commitment public key $pk = (p_B, y_B, g, h_2, y_B)$ and the trapdoor $td = x_B$.

2) $TC_{\text{Com}}$: To commit to a string $r$ with arbitrary length, the sender sends the commitment $\mathbf{r} = g^{h_2(r)}y_B^{x_B} \mod p_B$, where $r \in_R \mathbb{Z}_{q_B}$ is a randomness, and stores $\mathbf{dec} = t$.

3) $TC_{\text{Dec}}$: To decommit $\mathbf{r}$, the sender reveals $(r, t)$, so that the receiver can check if $\mathbf{r} \equiv g^{h_2(r')}y_B^{x_B} \mod p_B$.

4) $TC_{\text{Sign}}$: Given an answer $(r, t)$ to a commitment $\mathbf{com} = \mathbf{r}$, by using the trapdoor $x_B$, the receiver Bob can decommit $\mathbf{r}$ w.r.t. any string $r'$ by revealing $(r', t')$, where $t' = x_B^{-1}(h_2(r') - h_2(r')) + t \mod q_B$. It is easy to see that $(r', t')$ is also a valid answer to $\mathbf{r}$.

Note that the above DL-based trapdoor commitment scheme perfectly matches the Diffie–Hellman key setting. Namely, if a receiver has such a key pair for Schnorr signature, ElGamal signature, or DSA, etc., his key pair implicitly defines a secure trapdoor commitment without running any extra algorithm.

IV. THE PROPOSED PROTOCOL

In this section, we describe our new contract-signing protocol based on the RSA signature [42]. The basic idea is that Alice first splits her private key $d$ into $d_1$ and $d_2$ so that $d = d_1 + d_2 \mod \phi(n)$, as Park et al. did in [40]. Then, only $d_2$ is delivered to the TTP, while Alice keeps $(d, d_1, d_2)$ as secrets. To exchange her signature $\sigma_A = h(m)^d \mod n$ with Bob, Alice first sends partial signature $\sigma_1 = h(m)^{d_1} \mod n$ to Bob, and proves that $\sigma_1$ is prepared correctly in an interactive zero-knowledge way by exploiting Gennaro et al.’s protocol [27]. Moreover, to fully achieve abuse-freeness, this interactive zero-knowledge protocol is enhanced by a trapdoor commitment scheme (see Section III), which depends on Bob’s signature public key. After that, Bob sends his signature $\sigma_B$ on message $m$ to Alice, since he has been convinced that even if Alice refuses to reveal the second partial signature $\sigma_2 = h(m)^{d_2} \mod n$, the TTP can do the same thing.

As usual [36, 46], we assume that the communication channel between Alice and Bob is unreliable, i.e., messages inserted into such a channel may be lost due to the failure of computer network or attacks from adversaries. However, the TTP is linked with Alice and Bob by reliable communication channels, i.e., messages inserted into such a channel will be delivered to the recipient after a finite delay.

A. Registration Protocol

To use our protocol for exchanging digital signatures, only the initiator Alice needs to register with the TTP. That is, Alice is required to get a long-term voucher $V_A$ from the TTP besides obtaining a certificate $C_A$ from a CA. To this end, the following procedures are executed.

1) Alice first sets an RSA modulus $n = pq$, where $p$ and $q$ are two $k$-bit safe primes, i.e., there exist two primes $p'$ and $q'$ such that $p = 2p' + 1$ and $q = 2q' + 1$. Then, Alice selects her random public key $e \in_R \mathbb{Z}^*_n$ and calculates her private key $d = e^{-1} \mod \phi(n)$, where $\phi(n) = (p - 1)(q - 1)$. Finally, Alice registers her public key with a CA to get her certificate $C_A$, which binds her identity and the corresponding public key $(n, e)$ together.

2) Alice randomly splits $d$ into $d_1$ and $d_2$ such that $d = d_1 + d_2 \mod \phi(n)$ by choosing $d_1 \in_R \mathbb{Z}^*_n$, and computes $e_1 = d_1^{-1} \mod \phi(n)$. At the same time, she generates a sample message-signature pair $(w, \sigma_w)$, where $w \in \mathbb{Z}_n^* \setminus \{1, -1\}$, $\gcd(|w|, \phi(n)) \leq p'q'$, and $\sigma_w = w^{d_1} \mod n$. Then, Alice sends $(C_A, w, \sigma_w, d_2)$ to the TTP but keeps $(d_1, d_2, e_1)$ secret.

3) The TTP first checks that Alice’s certificate $C_A$ is valid. After that, the TTP checks that the triple $(w, \sigma_w, d_2)$ is prepared correctly. If everything is in order, the TTP stores $d_2$ securely, and creates a voucher $V_A$ by computing $V_A = \mathbf{Sign}_{\text{TPP}}(C_A, w, \sigma_w)$. That is, $V_A$ is the TTP’s signature on message $(C_A, w, \sigma_w)$, which guarantees that the TTP can issue a valid partial signature on behalf of Alice by using the secret $d_2$.

We give some notes on the above registration protocol. To get her certificate from a CA, Alice has to prove that modulus $n$ is the product of two safe primes. This technical issue is addressed in [27]. Of course, step (1) can be omitted if Alice has obtained such a certificate before she registers with the TTP. To validate the correctness of the triple $(w, \sigma_w, d_2)$, the TTP needs to do the following. First, the TTP validates that $w$ is an element of order at least of $p'q'$ by checking that $w \in \mathbb{Z}_n^* \setminus \{1, -1\}$, and that both $\gcd(w-1, n)$ and $\gcd(w+1, n)$ are not prime factors of $n$ [27, Lemma 1]. Then, Alice is required to show that she knows the discrete logarithm of $\sigma_w$ to the base $w$ via a zero-knowledge protocol interactively or noninteractively (see [27, Sec. 4.3]). Finally, the TTP checks whether $w \equiv \langle \sigma_w w^{d_2} \rangle \mod n$. If all those validations pass, the TTP accepts $(w, \sigma_w, d_2)$ as a valid triple and creates the voucher $V_A$ for Alice.

Though the above registration protocol is a little complicated, we remark that this stage needs to be executed only once for a sufficiently long period, for example, one year. In this period, Alice can fairly sign any number of contracts with all potential parties. Furthermore, it seems reasonable in the real world to require users to first register with the TTP before they are served. The reason is that the TTP is usually unlikely to provide free service for settling disputes between users. Moreover, for enhancing efficiency, the sample message $w$ can be fixed as a constant, e.g., $w = 2$, as pointed out by Gennaro et al. [27]. Compared with schemes based on verifiably encrypted signatures [2, 4], [6], one disadvantage of our registration protocol is that the TTP needs to keep a distinct secret $d_2$ for each registered user. However, this shortcoming can be eliminated by
Alice: Initiator
\[
\sigma_1 = h(m)^{d_1} \mod n.
\]
Pick a randomness \( t \), set \( r = c^{e_1} \mod n \) and \( \bar{f} = \text{TCom}(r, t) \).
Send \((r, t)\) if \( c \equiv \sigma_1^{i_j} \sigma_i^w \mod n \).
If \( \sigma_B \) is valid, send \( \sigma_2 = h(m)^{d_2} \mod n \).

Bob: Responder
\[
\frac{C_A, V_A, \sigma_1}{C_A, V_A, \sigma_1}
\]
Pick \( i, j \in [1, n] \) and set \( c = \sigma_1^{i_j} \sigma_i^w \mod n \).
If \( r \equiv h(m)^{2i_j} \mod n \) and \( \bar{f} \equiv \text{TCom}(r, t) \), send \( \sigma_B \).
If \( h(m)^2 \equiv (\sigma_1 \sigma_2)^w \mod n \), accept \( \sigma_2 \). Otherwise, apply the TTP's help.

Fig. 1. Signature exchange protocol.

some simple techniques. For example, the TTP can encrypt each concatenation of \( d_2 \) and the corresponding user’s unique identifier by exploiting a secure symmetric-key encryption algorithm, and then stores the results into its database. To extract a user's \( d_2 \) later, the TTP only needs to decrypt the corresponding record using the unique symmetric key.

B. Signature Exchange Protocol

We assume that a contract \( m \) has been agreed between Alice and Bob before they begin to sign it. In addition, it is supposed that the contract explicitly contains the following information: a predetermined but reasonable deadline \( t \), and the identities of Alice, Bob, and the TTP. Our signature exchange protocol is briefly illuminated in Fig. 1, and further described in detail as follows.

1) First, the initiator Alice computes her partial signature \( \sigma_1 = h(m)^{d_1} \mod n \), and then sends the triple \((C_A, V_A, \sigma_1)\) to the responder Bob. Here, \( h(\cdot) \) is a graphically secure hash function.

2) Upon receiving \((C_A, V_A, \sigma_1)\), Bob first verifies that \( C_A \) is Alice’s certificate issued by a CA, and that \( V_A \) is Alice’s voucher created by the TTP. Then, Bob checks if the identities of Alice, Bob, and the TTP are correctly specified as part of the contract \( m \). If all those validations hold, Bob initiates the following interactive zero-knowledge protocol with Alice to check whether \( \sigma_1 \) is indeed Alice’s valid partial signature on contract \( m \).

   a) Bob picks two numbers \( i, j \in [1, n] \) at random, and sends a challenge \( c \) to Alice by computing \( c = \sigma_1^{i_j} \sigma_i^w \mod n \).

   b) After getting the challenge \( c \), Alice calculates the respondend \( r = c^{e_1} \mod n \), and then returns her commitment \( \bar{f} = \text{TCom}(r, t) \) to Bob by selecting a random number \( t \), where \( \text{TCom} \) is the commitment algorithm of a secure trapdoor commitment scheme which depends on Bob’s public key (refer to Section III for details).

   c) When the commitment \( \bar{f} \) is received, Bob sends Alice the pair \((i, j)\) to show that he prepared the challenge \( c \) properly.

   d) Alice checks whether the challenge \( c \) is indeed prepared correctly, i.e., \( c \equiv \sigma_1^{i_j} \sigma_i^w \mod n \). If the answer is positive, Alice decommits the commitment \( \bar{f} \) by revealing the respondend \((r, t)\) to Bob. With the knowledge of \((r, t)\), Bob accepts \( \sigma_1 \) as valid if and only if \( r \equiv h(m)^{2i_j} \mod n \) and \( \bar{f} \equiv \text{TCom}(r, t) \).

3) Only if \( \sigma_1 \) is Alice’s valid partial signature and the deadline \( t \) specified in contract \( m \) is sufficient for applying dispute resolution from the TTP, Bob sends his signature \( \sigma_B \) on contract \( m \) to Alice, since he is convinced that another partial signature \( \sigma_2 \) can be released by the TTP, in case Alice refuses to do so.

4) Upon receiving \( \sigma_B \), Alice checks whether it is Bob’s valid signature on message \( m \). If this is correct, she sends Bob the partial signature \( \sigma_2 \) by computing \( \sigma_2 = h(m)^{d_2} \mod n \). When Bob gets \( \sigma_2 \), he sets \( \bar{f}_A = \sigma_1 \sigma_2 \mod n \), and accepts \( \sigma_2 \) as valid if and only if \( h(m)^2 = \sigma_2^w \mod n \). In this case, Bob can recover Alice’s standard RSA signature \( \sigma_A \) on message \( m \) from \( \bar{f}_A \) (more details are provided later). If Bob does not receive the value of \( \sigma_2 \) or only receives an invalid \( \sigma_2 \) from Alice timely, he applies help from the TTP via the dispute resolution protocol before the deadline \( t \) expires (see Section IV-C).

The following are further explanations on the above signature exchange protocol.

First, the interactive protocol exploited in step (2) is essentially the confirmation protocol for RSA undeniable signatures by Gennaro et al. [27], with respect to the private key \((d_1, e_1)\) and the public key \((n, w, \sigma_1)\). Note that similar approaches are used to construct e-payment protocol [15] and certified e-mail system [5]. In [27], it is proved that a successful execution of this zero-knowledge protocol guarantees that \( \sigma_1 = \beta h(m)^{d_1} \mod n \), where \( \beta \in \{1, -1, \alpha_1, \alpha_2\} \) and \( \alpha_i \)'s \( i = 1, 2 \) denote the two nontrivial elements of order 2. In this case, Bob accepts \( \sigma_1 \) as valid and sends his signature \( \sigma_B \) on contract \( m \) to Alice in step (3), since he is convinced that another partial signature \( \sigma_2 \) can be revealed by either Alice or the TTP. After that, if Alice does not reveal the value of \( \sigma_2 \) or only sends invalid \( \sigma_2 \) to Bob for a reasonable long period before the deadline \( t \), Bob resorts to the TTP to get the correct value.
of $\sigma_2$. If Alice honestly reveals $\sigma_2 = h(m)^{d_2} \mod n$ to Bob in step (4), we have $h(m)^2 \equiv \tilde{\sigma}_A^2 \mod n$, i.e., $\tilde{\sigma}_A = \sigma_1 \sigma_2 \mod n$ is valid. In this situation, Bob can recover the correct value of $\sigma_A$ from $\sigma_A$ by using the following recovery algorithm:

a) set $\sigma_A = \tilde{\sigma}_A$, if $h(m) = \tilde{\sigma}_A \mod n$;

b) set $\sigma_A = -\tilde{\sigma}_A \mod n$, if $h(m) = -\tilde{\sigma}_A \mod n$;

c) get $\sigma_A$ by factoring $n$, else, i.e., $h(m) \not\equiv \pm \tilde{\sigma}_A \mod n$.

We describe how Bob can factor $n$ and then get the value of $\sigma_A$ in case (c), i.e., $h(m)^2 = \tilde{\sigma}_A^2 \mod n$ but $h(m) \not\equiv \pm \tilde{\sigma}_A \mod n$. Note that the equality $h(m)^2 = \tilde{\sigma}_A^2 \mod n$ implies that $\tilde{\sigma}_A = (\beta h(m)^d \mod n, \beta \in \{1, -1, \alpha_1, \alpha_2\}$. When $\beta = \pm 1$, corresponding to cases (a) and (b), Bob can easily find the value of $\sigma_A$. So we conclude that case (c) means $\tilde{\sigma}_A = \alpha_i h(m)^d \mod n, i = 1, 2$. Recall that $\phi(n) = 4p^e q^f$, so we have $\tilde{\sigma}_A = \alpha_i h(m)^d \mod n = \alpha_i h(m)^d \mod n$. Therefore, Bob can get the value of $\alpha_i$ by computing $\alpha_i \equiv \tilde{\sigma}_A h(m)^{-1} \mod n$. It is well known that with the knowledge of such a non-trivial element of order 2, Alice’s RSA modulus $n$ can be easily factored, i.e., $(\alpha_i - 1)$ and $(\alpha_i + 1)$ are the two prime factors of $n$. Consequently, Bob can get Alice’s private key $d$ by using an extended Euclidean algorithm, and then obtain the value $\sigma_A$ by computing $\sigma_A = h(m)^d \mod n$.

Based on the above discussion, we conclude that case (c) will no happen in the real world unless Alice wants to reveal her private key. That is, if Alice revealed $\sigma_1 = \alpha_i h(m)^{d_1} \mod n$ and $\sigma_2 = h(m)^{d_2} \mod n$, Bob will not only be able to recover her signature $\sigma_A$ on contract $m$, but also could derive her private key $d$ (and then forge signatures). So we ignore case (c) in the discussions hereunder after an implicit assumption that any user does not want to compromise his/her own private key.

Second, the trapdoor commitment enhances the security of the above zero-knowledge protocol that shows the validity of partial signature $\sigma_1$. Specifically, using a commitment scheme in the above protocol forces Bob to prepare the challenge correctly (otherwise, he cannot get the response $r$), and therefore Bob cannot forward the intermediate results to convince an outsider of the validity of $\sigma_1$ after execution of the zero-knowledge protocol. Using a trapdoor commitment scheme here even makes Bob unable to collude with one or more outsiders during the execution of this zero-knowledge protocol by generating the challenge $c$ collectively. More discussions on this issue will be given in Section VI, as this is the exact reason why our contract-signing protocol is abuse-free.

Finally, the trapdoor commitment scheme relies on Bob’s public key so it may need some extra parameters other than his standard public key. However, as we discussed in Section III, at least for the two most popular public keys based on RSA and discrete logarithm problems, we at most need some implicit default parameters. For some special public keys, if such extra parameters are necessary, we can assume that they are specified in Bob’s public key certificate. Anyway, note that in our protocol the responder Bob does not need to register with the TTP at all, though the initiator Alice needs to do so.

C. Dispute Resolution Protocol

If Bob has sent his signature $\sigma_B$ to Alice but does not receive the value of $\sigma_2$ or only receives an invalid $\sigma_2$ from Alice before the deadline $t$, then he sends the TTP $(C_A, V_A, m, \sigma_1, \sigma_B)$ to apply dispute resolution. Upon receiving Bob’s application, the TTP performs as follows:

1) The TTP first verifies whether $C_A, V_A$, and $\sigma_B$ are Alice’s valid certificate, voucher, and Bob’s signature on contract $m$, respectively. After that, the TTP checks whether the deadline $t$ embedded in $m$ expires, and whether Alice, Bob, and itself are the correct parties specified in $m$. If any validation fails, the TTP sends an error message to Bob. Otherwise, continue.

2) Then, the TTP computes $\sigma_2 \equiv h(m)^{d_2} \mod n$ and checks whether $h(m)^2 \equiv (\sigma_1 \sigma_2)^{2e} \mod n$. If this equality holds, the TTP sends $(m, \sigma_2)$ to Bob and forwards $(m, \sigma_B)$ to Alice. Otherwise, i.e., $h(m)^2 \not\equiv (\sigma_1 \sigma_2)^{2e} \mod n$, the TTP sends an error message to Bob.

In the following, we explain why our dispute resolution protocol works. Since the TTP sets $\sigma_2 \equiv h(m)^{d_2} \mod n$, we conclude that $h(m)^2 \equiv (\sigma_1 \sigma_2)^{2e} \mod n$ if and only if $\sigma_1 \equiv \beta h(m)^{d_1} \mod n, \beta \in \{1, -1, \alpha_1, \alpha_2\}$. That is, the TTP can determine whether Bob has sent a valid $\sigma_1$ to apply dispute resolution by checking $h(m)^2 \equiv (\sigma_1 \sigma_2)^{2e} \mod n$. If this equality holds, the TTP reveals the correct value of $\sigma_2$ to Bob and forwards Bob’s signature $\sigma_B$ on contract $m$ to Alice. After getting the correct $\sigma_2$, Bob can recover Alice’s signature $\sigma_A$ on contract $m$ by employing the recovery algorithm given in Section III. In the case of $h(m)^2 \not\equiv (\sigma_1 \sigma_2)^{2e} \mod n$, the TTP knows that Bob is a cheater, and so only sends an error message to him.

Note that if the $\sigma_1$ sent to the TTP is prepared as $\sigma_1 = \alpha_i h(m)^{d_1} \mod n$, the TTP can also get Alice’s private key $d$ as Bob does.

Remark 1: Deadline $t$ is a very important parameter in our protocol. If Bob receives valid $\sigma_1$ at a time which is very close to the deadline $t$, he should not reveal his signature $\sigma_B$ to Alice. In this situation, Bob could have several choices to guarantee fairness: 1) ignore this protocol instance; 2) get valid $\sigma_2$ from the TTP directly by initiating dispute resolution protocol; or 3) require Alice use a new deadline $t'$ and run the signature exchange protocol with Alice again.

V. SECURITY DISCUSSION

Based on the descriptions and discussions presented in Section IV, we know that in the normal situation, i.e., both involved parties are honest and the communication channel is in order, each of the two parties can get the other’s signature on the same contract correctly, and the TTP is not involved. In other words, our scheme is complete and optimistic.

Now, we discuss the abuse-freeness. First, after the execution of the zero-knowledge protocol in Step (2) of the Signature Exchange Protocol, if Bob forwards the partial signature $\sigma_1$ with the proof $(c, r, i, j, l, t)$ to others, nobody (other than Alice and the TTP) believes that $\sigma_1$ is indeed Alice’s partial signature on contract $m$. Here are the reasons. For any contract $m$, Bob himself can simulate such a proof for any purported $\sigma_1$, which may be valid or invalid with respect to contract $m$, as follows: By first choosing three random numbers $i$, $j$, and $l$, Bob can then set $c = \sigma_1^2 \tilde{\sigma}_B \mod n, r = h(m)^{2i w} \mod n$, and $\tilde{r} = T \text{Com}(r, t)$. Furthermore, such a simulated proof is
computationally indistinguishable from the real proof, i.e., the
authentic transcript generated by the interaction between Alice
and Bob via running the signature exchange protocol given in
Fig. 1. Therefore, if Bob forwards the transcript \((s, r, i, j, r, t)\)
to an outsider Charlie after the execution of the zero-knowledge
protocol for validating partial signature \(\sigma_1\), Charlie cannot
accept this as convincing evidence showing the validity of \(\sigma_1\),
since Charlie (and any user) knows that such a transcript could
be simulated by Bob alone. (In fact, this is called zero-knowledge
property as what Bob gets via running the protocol is just
something he can compute without Alice’s interaction, i.e., Bob
obtains nothing or zero-knowledge except the confirmation that
\(\sigma_1\) is valid.) So, the proposed protocol is abuse-free after the
execution of the zero-knowledge protocol.

Note that, however, if we used a standard commitment scheme
rather than a trapdoor commitment scheme, Bob is still
able to convince an outsider Charlie that \(\sigma_1\) is a valid partial
signature by colluding with Charlie during the execution of the zero-knowledge protocol. To this end, Charlie and Bob first
independently compute two challenges
\[
\begin{align*}
\sigma_1 &= \sigma_1^{(2 i + j) \mod n} \\
\sigma_2 &= \sigma_2^{(i + j) \mod n}
\end{align*}
\]
respectively, where random number pairs \((i_1, j_1) \in \mathbb{R}_{[1, n]} \times \mathbb{R}_{[1, n]}\) are chosen by them separately. Then, they combine these two challenges as
one by setting \(c = c_1 \cdot c_2 = \sigma_1^{(2i+j)} \cdot \sigma_2^{(i+j)} \mod n\). After
that, as the verifier, Bob runs the zero-knowledge protocol with
Alice honestly to get a commitment \(\varphi\) and the answer \((r, t)\) for
\(\varphi\) by revealing \((i = i_1 + i_2, j = j_1 + j_2)\). By first informing
Charlie the value of \(\varphi\) and then asking the values of \((i_1, j_1)\), Bob
can convince Charlie that \(\sigma_1\) is Alice’s valid partial signature,
since nobody can change the answer \((r, t)\) for the commitment
\(\varphi\) even after seeing the values of \((i, j)\).

In contrast, as we exploit a trapdoor commitment scheme TC
to hide Alice’s real response \(r\), the above collusion attack does not work any more. The reason is that even \(\sigma_1\) is not Alice’s
valid partial signature; Bob is able to run the above attack with
Charlie successfully without any interaction with Alice as follows.
After \(c_1\) and \(c_2\) are released, Bob first selects two random numbers \((r, t)\) to make a commitment \(\varphi = TC\text{com}(r, t)\). After
forwarding \(\varphi\) to Charlie, Bob can get \((i_1, j_1)\) which allows him to
compute \(i = i_1 + i_2\) and \(j = j_1 + j_2\) and then find a number \(t'\)
for the value \(\varphi' = h((n)^{2j+t'}) \mod n\), thanks to the TC\text{com}\$\$ algorithm of our trapdoor commitment scheme. Finally, Bob returns
the simulated but valid answer \((r', t')\) to Charlie, who is unable to
tell whether this is a true response from Alice or a simulated
answer from Bob by using his secret key. This means that our
contract-signing protocol is also abuse-free during the execution
of zero-knowledge protocol.

Therefore, the proposed contract-signing protocol fully satis-
fies the abuse-freeness either after execution or during the exe-
cution of the zero-knowledge protocol in Step (2) of the Signa-
ture Exchange Protocol.

Moreover, our protocol overcomes the security flaw in Park et al.’s scheme. Namely, if Alice is honest, the TTP cannot
derive Alice’s private key \(d\) from \(d_2\) and other public infor-

\[\text{2In the early version of this paper [45], only a (standard) commitment scheme is specified, so the protocol is vulnerable to this attack. Due to this reason, we update our protocol here by explicitly stressing that a trapdoor commitment scheme is necessary in the proposed contract-signing protocol to fully achieve abuse-freeness.} \]
rarameter $k$). This means that nobody can generate valid $\sigma_1$ except Alice, and that nobody can generate valid $\sigma_2$ except Alice and the TTP.

Case (1) implies that in step (1) of our signature exchange protocol, Alice first properly computes $\sigma_1 = h(m)^d_1 \mod n$, and sends the triple $\{C_A, V_A, \sigma_1\}$ to Bob, where $C_A$ is Alice’s public key certificate issued by a trusted CA, and $V_A$ is Alice’s valid voucher created by the TTP. The purpose of step (2) in our signature exchange protocol is that Alice interactively convinces Bob to accept valid $\sigma_1$ in a zero-knowledge proof way. According to [27, Th. 1], we know that even if Bob cheats in any possible way, he cannot learn other information except $\sigma_1$ is valid, i.e., $\sigma_1 = \beta h(m)^{d_1} \mod n$, for some $\beta \in \{1, -1, \alpha_1, \alpha_2\}$. Actually, $\beta$ must be 1 since Alice is honest in this setting. This also implies that Bob cannot factor Alice’s RSA modulus $n$ by first getting a nontrivial element of order 2.

Upon receiving the valid value of $\sigma_1$, Bob has to make a choice whether he should send his signature $\sigma_B$ on contract $m$ to Alice. If Bob does, honest initiator Alice returns back her second partial signature $\sigma_2 = h(m)^{d_2} \mod n$ as Bob expects. In such a situation, Bob gets Alice’s signature on contract $m$ by setting $\sigma_A = \sigma_1 \sigma_2 \mod n$, while Alice also obtains Bob’s signature $\sigma_B$ simultaneously. If Bob does not send $\sigma_B$ or only sends an incorrect $\sigma_B$ to Alice, he cannot get the value of $\sigma_2$ from Alice in step (4). Furthermore, in this setting, Bob also cannot get the value of $\sigma_2$ from the TTP so that Alice does not obtain his signature $\sigma_B$. The reason is that in our dispute resolution protocol, to get the value of $\sigma_2$ from the TTP, Bob has to submit valid $\sigma_1$ and $\sigma_2$ to the TTP. Once those values are submitted, Bob indeed gets $\sigma_2$ from the TTP but Alice receives $(m, \sigma_B)$ from the TTP, too. Therefore, once again, Bob and Alice get the other’s signature on contract $m$ at the same time.

Case 2: Bob is honest, but Alice is cheating. In our signature exchange protocol, Alice may cheat in any or some of the following steps: step (1), step (2), and step (4). First of all, according to the specification of our signature exchange protocol, to get the signature $\sigma_B$ on contract $m$ from the honest responder Bob, the initiator Alice has to convince Bob accepting $\sigma_1$ as a valid partial signature in step (2). Recall that step (2) is exactly Gennaro et al.’s confirmation protocol for RSA undeniable signatures, and that their protocol satisfies the property of soundness [27, Th. 1]. The soundness means that the possible cheating Alice (prover), even computationally unbounded, cannot convince Bob (verifier) to accept an invalid $\sigma_1$ as valid with non-negligible probability. Therefore, we conclude that to get $\sigma_B$ from Bob, Alice has to send valid $\sigma_1$ (with valid $C_A$ and $V_A$) in step (1) and perform honestly in step (2). In other words, Alice has to send $\sigma_1 = \beta h(m)^{d_1} \mod n$ to Bob unless she does not want to get Bob’s signature $\sigma_B$, where $\beta \in \{1, -1, \alpha_1, \alpha_2\}$.

According to our discussions given in Section IV, we know that Alice is not so silly by preparing and sending $\sigma_1 = \alpha h(m)^{d_1} \mod n$ to Bob. Otherwise, Bob can drive her private key $d$ (and then compute signature $\sigma_A$), though she indeed can get Bob’s signature $\sigma_B$. Therefore, to get signature $\sigma_B$ from Bob, Alice has to compute $\sigma_1 = \pm h(m)^{d_1} \mod n$ and send it to Bob. In this situation, Bob receives valid $\sigma_1 = \pm h(m)^{d_1} \mod n$ from Alice before Alice gets valid $\sigma_B$ from Bob. After that, step (4) is the only one possible cheating chance for Alice, i.e., she may refuse to reveal $\sigma_2$ or just send an incorrect $\sigma_2$ to Bob. However, this cheating behavior does not harm Bob essentially, since he can get the value of $\sigma_2$ from the TTP via our dispute resolution protocol. The reason is that Bob has received valid $\sigma_1$ before he sends $\sigma_B$ to Alice. After getting the value of $\sigma_2$ from the TTP, Bob can recover Alice’s signature $\sigma_A$ according to the recovery algorithm specified in Section III-B. Therefore, in case (b) where Bob is honest but Alice is dishonest, Alice cannot get Bob’s signature such that Bob does not obtain her signature.

Based on the above analysis, we conclude that the proposed protocol is not advantageous to any dishonest party. In other words, our contract-signing protocol satisfies the property of fairness.

VI. EFFICIENCY

Table I shows the comparison of efficiency between our new protocol and several other RSA-based solutions, i.e., Asokan et al.’s scheme [2], [3] from verifiable escrow, Atienese’s scheme [4] from verifiably encrypted signature, and Park et al.’s scheme [40] from multisignature. In the comparison, we analyze the overheads of computation and communication in the signature exchange protocol needed by both Alice and Bob in the normal case. In other words, the operations of the dispute resolution protocol are not discussed here. Moreover, we take the number of modular exponentiations as the computational cost since exponentiation is the most expensive cryptographic operation in the finite field $\mathbb{Z}_n$. In addition, note that a modular exponentiation in $\mathbb{Z}_n$ requires about $1.5 \times |n|$ modular multiplications, and that exponentiation of the form $a_1^{1/2} a_2^{1/2}$ is only equivalent to 1.167 single exponentiation by means of an exponent array [38, p. 618].

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>75</td>
<td>13.3</td>
<td>7</td>
<td>10.5</td>
<td>600</td>
</tr>
<tr>
<td>8000</td>
<td>916</td>
<td>600</td>
<td>1336</td>
<td></td>
</tr>
</tbody>
</table>

TABLE I

Efficiency Comparison

For comparison, we make similar but different assumptions from [4] and [40]. Namely, we assume that the length of RSA modulus $n$ is 1200 bit, and that the hash function $h(\cdot)$ has 160-bit fixed output. For simplicity, we also assume that $\sigma_B$ could be generated and verified by one modular exponentiation separately, and that the voucher $V_A$ can be validated by one modular exponentiation, too. However, the overhead related to Alice’s certificate $C_A$ is excluded, as in [4] and [40], since such validation may be as simple as to check the certificate list on the CA’s web site.

Some numbers listed in Table I are different from the results that appeared in [4] and [40], since we take into consideration all exponentiations needed in the signature exchange protocols by...
both Alice and Bob, while Anteniese only concerned the amount of each signature algorithm, and Park et al. [40] only considered the overhead required for creating/verifying the fairness primitives (i.e., $\sigma_1$ and $V_A$). For example, Anteniese did not include the overheads of creating and checking the proof for proving the equality of two discrete logarithms, while Park et al. did not estimate the overheads of generating and verifying Alice’s signature $\sigma_A$. Our analysis is more reasonable since it accurately reflects what happens in practice. In addition, note that the numbers for the Asokan et al.’s scheme were taken from [40] directly.

According to the results in Table I, the computational efficiency of our scheme is in the middle between Park et al.’s scheme and Anteniese’s scheme, while the communication cost of our scheme increases by 123% and 46% more than that of Park et al.’s scheme and Anteniese’s scheme, respectively. The overhead of communication becomes larger naturally, since our scheme exploits interactive protocol to prove the validity of $\sigma_1$. The bonus in our new scheme is that Bob cannot show the validity of $\sigma_1$ to other parties, i.e., abuse-freeness, as we discussed before. We believe that this cost deserves the advantage of our scheme in the situations where the intermediate results should not be revealed unfairly. Actually, all three schemes are suited for most applications where the cost of communication is not the main concern.

VII. Conclusion

In this paper, based on the standard RSA signature scheme, we proposed a new digital contract-signing protocol that allows two potentially mistrusted parties to exchange their digital signatures on a contract in an efficient and secure way. Like the existing RSA-based solutions, the new protocol is fair and optimistic, i.e., two parties get or do not get the other’s digital signature simultaneously, and the TTP is only needed in abnormal cases that occur occasionally. However, different from all previous RSA-based contract-signing protocol, the proposed protocol is further abuse-free. That is, if the contract-signing protocol is executed unsuccessfully, each of the two parties cannot show the validity of intermediate results generated by the other party to outsiders, during or after the procedure where those intermediate results are output. In other words, each party cannot convince an outsider to accept the partial commitments coming from the other party. This is an important security property for contract signing, especially in the situations where partial commitments to a contract may be beneficial to a dishonest party or an outsider. Technical details are provided to show that our protocol meets a number of desirable properties, not only those just mentioned.

In addition, exploiting some techniques of Park et al. [40], our protocol can be adapted to fair payments in e-commerce (though their solution has a security flaw). In this setting, one customer purchases digital goods from a merchant via the Internet by paying with a digital check or cash. The extended scheme could implement such an electronic transaction between two parties fairly. That is, it is guaranteed that the customer gets the digital goods from the merchant if and only if the merchant gets the money from the customer.

Finally, using the technique of threshold RSA signature introduced by Shoup [44], the proposed protocol could be extended for the scenarios where the trust on a single TTP needs to be distributed into multiple TTPs, or a contract is required to be signed only by a given quota of members cooperatively.

REFERENCES


J. Rompel, “One-way functions are necessary and sufficient for secure signatures,” in Proc. STOC’90, 1990, pp. 387–394, ACM.


Guilin Wang received the PhD. degree in computer science from the Institute of Software, Chinese Academy of Sciences, China, in March 2001.

He is currently a Lecturer at the School of Computer Science, University of Birmingham, U.K. Before this, he was a Research Scientist at the Institute for Infocomm Research (I2R), Singapore (June 2002 to September 2007), and an Assistant Professor at the Institute of Software, Chinese Academy of Sciences (March 2001 to May 2002). He has served as a program co-chair for two international security conferences (ICICS’08 and ISA’09), a program committee member for more than 30 international information security related conferences or workshops, and a reviewer for more than 20 international journals. To date, he has more than 50 technical publications in the areas of applied cryptography, information security, and electronic commerce. In particular, he is interested in the analysis and design of digital signatures and fair exchange protocols. His homepage is http://www.cs.bham.ac.uk/~gzw/.