5-2015

# Electronic Contract Signing without using Trusted Third Party

Zhiguo WAN
*Singapore Management University*, zgwan@smu.edu.sg

Robert H. DENG
*Singapore Management University*, robertdeng@smu.edu.sg

David Kuo Chuen LEE
*Singapore Management University*, davidlee@smu.edu.sg

# Electronic Contract Signing Without Using Trusted Third Party

Zhiguo Wan[1(✉)], Robert H. Deng[2], and David Lee[1]

[1] Sim Kim Boon Institute for Financial Economics, Singapore Management University, Singapore, Singapore
zgwan@smu.edu.sg
[2] School of Information Science, Singapore Management University, Singapore, Singapore

**Abstract.** Electronic contract signing allows two potentially dis-trustful parties to digitally sign an electronic document "simultaneously" across a network. Existing solutions for electronic contract signing either require the involvement of a trusted third party (TTP), or are complex and expensive in communication and computation. In this paper we propose an electronic contract signing protocol between two parties with the following advantages over existing solutions: 1) it is practical and scalable due to its simplicity and high efficiency; 2) it does not require any trusted third party as the mediator; and 3) it guarantees fairness between the two signing parties. We achieve these properties by employing a trustworthy timestamping service in our protocol, where the timestamping service can be either centralized or decentralized. We also provide a detailed analysis on security and performance of our scheme.

## 1 Introduction

Contract signing is a frequent activity in business as well as in our daily lives, e.g. purchasing real estates and insurance, employment, and financial trading. Almost all important commercial and financial activities require legally signed contracts to guarantee that the involved parties commit to mutually agreed terms and conditions and fulfill their obligations. Most important contracts today, however, are signed physically, e.g., using pen and paper. As a result, signing a contract could be very time consuming and costly, and it may take several days or weeks to complete. As the information technology and digital signature laws are becoming more and more pervasive, it is high time to consider digitally signing electronic contracts across the Internet.

Fairness is a critical requirement for electronic contract signing, which ensures that the two involved parties either obtain each other's signature "simultaneously" or nothing useful. The fairness property implies that a dishonest party who tries to cheat cannot get any advantage over the other party. Achieving fairness is straightforward for physically signing contracts; however, due to lack of simultaneity in computer networks, achieving true fairness in electronic contract signing over Internet has been a challenging problem. Many electronic contract


M. Qiu et al. (Eds.): NSS 2015, LNCS 9408, pp. 386–394, 2015.
DOI: 10.1007/978-3-319-25645-0_27

signing or fair exchange protocols have been proposed in the literature, but none of them have gained widespread adoption. Among these protocols, some require an online or offline trusted third party (TTP) to mediate the exchange, while others are TTP-free but impractical due to high computation and communication overhead.

In this paper, we propose a practical electronic contract signing protocol built on a trustworthy timestamping service, without using any TTP. Our protocol enjoys the following advantages over existing ones. First, it is very practical and scalable due to its simplicity and high efficiency in computation and communication. Unlike TTP-free protocols which need multiple rounds of communication between the two signing parties [1], our protocol only requires three messages to be exchanged. The protocol at the same time does not require complex computations except a few digital signatures, which makes it very efficient in computation. Secondly, our protocol does not require any TTP, thus removing the single point of failure and the bottleneck for scalability. Third, the protocol guarantees fairness between the two signing parties. The two parties negotiate and agree on a deadline before which the contract must be signed by both of them. Failing to fulfill this requirement leads to invalidity of the contract, and this protects the interest of the first party against malicious behaviors of the second party.

## 2 Related Work

Electronic commerce has greatly motivated research in electronic contract signing. According to the extent to which a TTP is involved in the contract signing process, existing protocols can be divided into three groups: 1) TTP-free protocols; 2) protocols with an online TTP; and 3) protocols with an offline TTP.

**TTP-Free Protocols.** TTP-free protocols have the advantage of no need for a specialized TTP, and its design goal is to fulfill the computational fairness [1]. The idea is for two signing parties to exchange their signatures on a contract "bit-by-bit". Whenever any of the two parties terminates prematurely, both of them can still complete the exchange offline by exhaustively searching the remaining bits of the signatures. Although this approach enjoys the great advantage of being TTP-free, it is impractical for most real-world applications. The main reason is due to the high computation and communication cost of such protocols.

**Online-TTP Protocols.** Since a TTP facilitates the execution of the signing process, contract-signing protocols with an online TTP can be much simpler and efficient [2]. Under the online-TTP setting, a TTP acts as a mediator between the two signing parties. The main issue with the online-TTP protocols is that the TTP is likely to become a performance and security bottleneck of the system, especially when there are a huge number of participants in the system.

**Offline-TTP Protocols.** Contract-signing protocols with an offline TTP [3–5] are more appealing and practical because the TTP is not involved during the execution of an exchange unless some problems occur. Fair exchange protocols

for digital signatures employed two different cryptographic techniques: verifiable encrypted signatures [5] and verifiable escrows [3].

## 3 The Proposed Protocol

### 3.1 System Model

Our proposed contract signing protocol involves three types of generic entities or roles: a PKI, a timestamping server and signing parties. The design goal is to enable any two parties to sign an electronic contract in a fair manner. We assume that each party has a public key/private key pair in a signature scheme. The PKI is responsible for public key certificate management for all parties. To ensure security and resilience of the PKI, a resilient PKI system in [6] can be used.

The trustworthy timestamping service is crucial to ensure fairness in our contract signing protocol. It is used to produce irrefutable timestamped proofs that digital signatures on a common contract document are submitted by the two parties before a mutually agreed deadline. These proofs are maintained by the timestamping server, and no party can forge or tamper the proofs without being detected.

### 3.2 Threat Model

We assume that both the PKI and the timestamping service are trustworthy. Specifically, the timestamping server does not need to be trusted as a TTP.

The adversary can be a peer with whom a party wants to sign a contract, other uninvolved parties or simply an outsider. The adversary is assumed to have only limited computation capability, and he cannot break the digital signature algorithm used in the proposed protocol. The ultimate goal of the adversary is to trick an honest party into signing a contract he would otherwise not want to sign. The adversary can launch both passive and active attacks against the contract signing protocol, including message eavesdropping, modification, forgery, replaying and so on.

### 3.3 Protocol Description

After two parties, referred to here as $A$ and $B$, have finished negotiating terms and conditions, they agree on a final contract document $\mathcal{C}$. Let $Cert_X$ denote the public key certificate of party $X$ and $\mathsf{Sign}_X(M)$ the signature generated by signing message $M$ with the private key of party $X$, where $X$ can be either $A$ or $B$. Let $\mathsf{Ver}_X(M)$ denote verification of a signature on $M$ using $X$'s public key, which outputs true if the verification is successful and false otherwise.

As showed in Fig. 1, the proposed protocol is composed of three phases: signing by the first party, verification and signing by the other party, and timestamping the signed contract. They are described in detail as follows.

**Signing by the First Party.** To sign a contract, $A$ and $B$ negotiate with each other and decide on a future time $T_d$ as the deadline before which the two parties must sign on $\mathcal{C}$; otherwise the contract becomes void. How to set this parameter depends on specific applications. It may be in seconds, minutes, hours, or even days. Party $A$, who signs the contract first, needs to make sure that this parameter is not too long to render herself in a disadvantageous situation. For example, if $A$ wants to trade forex with the other party, then $T_d$ should be as short as a few seconds since forex prices fluctuate at seconds.

Party $A$ then creates a message containing a hash of the contract $\mathcal{C}$, the deadline $T_d$, $A$'s identity and $B$'s identity, and signs this message with her private key. That is, $A$ generates the signature $Sig_A = \mathsf{Sign}_A(H(\mathcal{C})|T_d|A|B)$, where $|$ denotes concatenation, and $H()$ is a secure one-way hash function. Then $A$ sends the following to $B$:

$$H(\mathcal{C}), T_d, Sig_A, Cert_A \tag{1}$$

**Verification and Signing by the Second Party.** Upon receiving the message from $A$, the second party, $B$, first checks that the $Cert_A$ is valid, the deadline $T_d$ has not expired and there is sufficient time left to finish the contract; otherwise he aborts the protocol. To continue the contract signing protocol, $B$ checks that $H(\mathcal{C})$ is the hash computed over the contract document $\mathcal{C}$ he negotiated with $A$, and verifies that $Sig_A$ is $A$'s valid signature on $(H(\mathcal{C})|T_d|A|B)$ . If the verifications fail, $B$ aborts; otherwise, $B$ signs the message $(H(\mathcal{C}), T_d, A, B, Sig_A)$ to obtain a signature $Sig_B = \mathsf{Sign}_B(H(\mathcal{C})|T_d|A|B|Sig_A)$, and sends the following message to the timestamping server:

$$H(\mathcal{C}), T_d, A, B, Sig_A, Sig_B, Cert_A, Cert_B. \tag{2}$$

We refer to the above message $(H(\mathcal{C}), T_d, A, B, Sig_A, Sig_B, Cert_A, Cert_B)$ as the signed contract and denote it as **CT** in the following.

**Timestamping the Signed Contract.** After receiving **CT** from $B$, the timestamping server **TS** first checks that $Cert_A$ and $Cert_B$ are valid, that $T_d$ has not expired and that the two signatures $Sig_A$ and $Sig_B$ were generated on the same hash value $H(\mathcal{C})$ by parties $A$ and $B$, respectively. Only if all verifications are successful will **TS** proceed to generate a trustworthy timestamp on the signed contract $\mathbf{CT} = (H(\mathcal{C}), T_d, Sig_A, Sig_B, Cert_A, Cert_B)$. The timestamping server then sends the following message to both parties $A$ and $B$ as a proof that the contract has been signed successfully:

$$\mathbf{CT}, \mathbf{ts}(\mathbf{CT}), \tag{3}$$

where $\mathbf{ts}(\mathbf{CT})$ denotes the timestamp on **CT**. What constitutes the timestamp will be described in the following subsection.

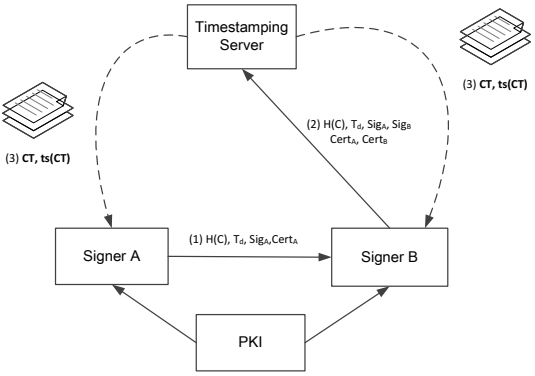



**Fig. 1.** The Contract Signing Protocol.

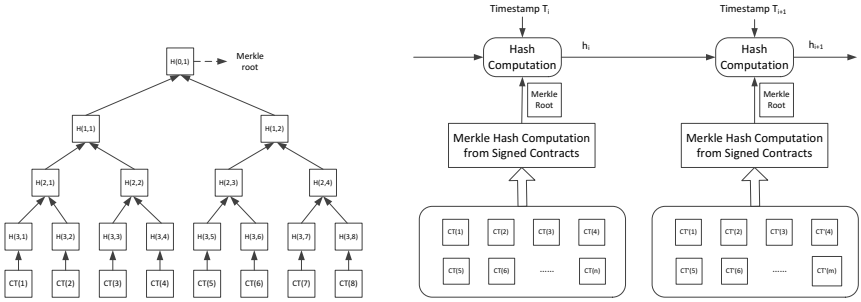
### 3.4 Timestamping Service

The trustworthy timestamping service referred to above in our protocol description can be implemented with a centralized server as in [7] or a decentralized system based on the blockchain mechanism in Bitcoin [8]. Both implementations adopt a similar approach to trustworthy timestamping as described below.

This timestamping service is built on the Merkle tree structure, a binary tree as showed in 2(a). A node at $(i, j)$ on the Merkle tree has two children: $(i + 1, 2j - 1)$ and $(i + 1, 2j)$. The root node is labeled as $(0, 1)$, meaning it is the first node at layer 0. The value $H(i, j)$ for a leaf node $(i, j)$ is simply the hash of the corresponding contract content, i.e. $H(\mathbf{CT}(i))$. The value of an interior node is calculated from its two children by the formula $H(i, j) = H(H(i+1, 2j-1)|H(i+1, 2j))$. The hash function can be any cryptographically secure hash algorithm, e.g. SHA256. The value calculated at the root is called the Merkle root.

The timestamping service divides time into fixed time intervals and at the end of each interval compresses submitted contracts into a Merkle root. This Merkle root, along with the current time, is taken as an input into a hash chain which is computed and maintained by a server in the centralized system or by a group of nodes in the decentralized system. The output of the hash chain and some auxiliary data corresponding to the specific contract are returned to the parties as the trustworthy timestamp on their contract.

The process of timestamping a Merkle root and hence all the contracts embedded in the root proceeds as illustrated in Fig. 2(b). In each fixed time interval, a Merkle root is calculated as above for all contracts submitted during this interval. The Merkle root, the current time and the previous hash value of the hash chain are fed into the hash function to calculate a new value, and the hash chain is extended by 1. Suppose the current time is $T_i$, the previous hash value of the hash chain is $h_{i-1}$, and the current Merkle root is $root_i$, then the new hash value is computed as $h_i = H(h_{i-1}|T_i|root_i)$.




(a) The Merkle Hash Tree (b) Timestamping the Contracts

**Fig. 2.** The Timestamping Service

The timestamp for a contract, say $\mathbf{CT}(3)$, at time interval $T_i$, is given by

$$\mathbf{ts}(\mathbf{CT}(3)) = (H(0,1), H(1,2), H(2,1), H(3,4), T_i, h_{i-1}, h_i).$$

For a centralized solution, the timestamping server needs to publish all tuples $(h_i, T_i, root_i)$. The latest hash values obtained in this way by a centralized server need to be published on an unalterable public media in a fixed interval (e.g., New York Times every week [7]). Therefore, even the centralized server cannot change the hash chain or the Merkle tree after publishing the last value on the hash chain. Thus the centralized server needs not be a TTP.

A decentralized timestamping service can be realized based on the notion of blockchain in Bitcoin [8], where hash values represent proof-of-work done collectively by a group of entities (e.g., users of the timestamping service). At each time interval $T_i$, the group of entities finds a random value as the input to the hash chain computation formula such that the output of the hash computation is less than a pre-determined public threshold. That is, the hash chain computation formula becomes $h_i = H(h_{i-1}|T_i|root_i|r_i)$, and one has to find an $r_i$ such that $h_i$ is less than the public threshold, which is called proof-of-work.

Finally, all hash values and the proof-of-work $r_i$'s are published on the blockchain so that everyone can check them. Due to the one-wayness of the hash function $H()$, finding a valid $r_i$ such that $h_i$ is less than a threshold could be very difficult. The amount of computation to find $r_i$ can be adjusted by varying the value of the threshold.

Since finding a valid $r_i$ is so difficult and the hash chain is continuously increasing, it is computationally infeasible to change a timestamped contract which is embedded in the hash chain, especially when the chain becomes longer [8]. Hence this decentralized system achieves trustworthy timestamping without relying on any centralized entity or trusted third party.

### 3.5 Timestamp and Contract Verifications

When both parties receive a timestamp on their signed contract from the timestamping server, they proceed to verify the timestamp for correctness. Without

loss of generality, assume that the centralized timestamping service in [7] is used. We describe the verification process by taking the aforementioned timestamp $\mathbf{ts}(\mathbf{CT}(3)) = (H(0,1), H(1,2), H(2,1), H(3,4), T_i, h_{i-1}, h_i)$ as an example. To verify this timestamp, both parties first check that the Merkle root is indeed computed from $\mathbf{CT}(3)$ by evaluating the hash tree from bottom to the root. That is, each party computes and checks if the following equations hold:

$$\begin{aligned} H(3,3) &= H(\mathbf{CT}(3)); \\ H(2,2) &= H(H(3,3)|H(3,4)); \\ H(1,1) &= H(H(2,1)|H(2,2)); \\ root &= H(H(1,1)|H(1,2)); \\ root &\stackrel{?}{=} H(0,1). \end{aligned}$$

Then each party verifies if the hash chain value is correctly calculated by testing the following equation: $h_i \stackrel{?}{=} H(h_{i-1}|T_i|root)$.

Next, both parties retrieve subsequent published tuples $(h_{i+1}, T_{i+1}, root_{i+1})$, $(h_{i+2}, T_{i+2}, root_{i+2})...$, $(h_m, T_m, root_m)$ from the timestamping server, where $h_m$ is the latest hash value published on the unalterable public media (i.e. newspaper). Then they verify all of the hash values are correctly computed as follows: $h_{i+k} \stackrel{?}{=} H(h_{i+k-1}|T_{i+k}|root_{i+k})$, where $k = 1, 2, ..., m-i$. If all verifications are successful, then they can be ensured that the timestamped proof is trustworthy.

Whenever a dispute occurs about a contract, either party can present the original contract document $\mathcal{C}^*$, the signed contract $\mathbf{CT} = (H(\mathcal{C}), T_d, A, B, Sig_A, Sig_B, Cert_A, Cert_B)$ and its timestamp to a judge for dispute resolution. To verify that parties $A$ and $B$ had indeed committed to the contract before time $T_i$, the judge first tests if the following equations are true:

$$\begin{aligned} H(\mathcal{C}^*) &\stackrel{?}{=} H(\mathcal{C}); \\ Sig_A &\stackrel{?}{=} \mathsf{Sign}_A(H(\mathcal{C})|T_d|A|B); \\ Sig_B &\stackrel{?}{=} \mathsf{Sign}_B(H(\mathcal{C})|T_d|A|B|Sig_A). \end{aligned}$$

Then the judge verifies the timestamp following the same procedure as discussed earlier in this subsection. The contract is declared valid only if the signed contract and the timestamp both pass verification.

## 4 Discussion and Analysis

**The First Party is Dishonest.** In any contract signing scenario, the party who signs first is always in an unfavorable situation since he is the first to make a commitment. The second party always has two options to choose depending on which one is in his favor. The best choice for the first party $A$ is to set $T_d$ as early as possible so that party $B$ is left with little time to exploit the situation. Nevertheless, $A$ may try to cheat $B$ by modifying content of the contract, delaying to send her signature, or changing the deadline $T_d$. But none of these methods can bring $A$ any advantage over $B$.

**The Second Party is Dishonest.** As the last one to commit to the contract, the second party $B$ can always choose whether or not to commit the contract, and hence possesses obvious advantages over the first committer $A$. Specifically, a dishonest $B$ can launch attacks by modifying the deadline $T_d$, delaying contract signing, or modifying the contract content. But these attacks will be easily detected by the timestamping server.

**Contract Privacy.** Privacy of content of a signed contract is well protected from disclosure by hashing the contract instead of submitting the original contract document. As a result, even the timestamping server does not know the content, but only a hash over the contract. Only when a dispute occurs between the signing parties will the contract document need to be disclosed for dispute resolution.

**Performance.** The proposed scheme is very concise with only three message transmissions. The computation cost for the first party is only one signature generation, while the second party needs to verify the first party's signature and then generate his own signature. For each contract timestamping request, the timestamping server needs to verify two signatures, which is insignificant for a resourceful server. The computation cost of the timestamping server is on the order of $O(n)$ where $n$ is the number of requests for timestamping services, so the proposed protocol scales to the number of requests.

## 5 Conclusion

In this paper, we have proposed a practical fair electronic contract signing protocol which does not require any online or offline TTP. The proposed protocol relies on a trustworthy timestamping service which can be implemented without any trusted third party. The protocol contains only three messages and is very efficient in computation and communication. We have also provided a detailed analysis on its security and performance. Due to its simplicity and efficiency, the proposed protocol is highly practical for many e-commerce applications.

## References

1. Goldreich, O.: A simple protocol for signing contracts. In: Proc. CRYPTO 1983, pp. 133–136. Plenum Press (1983)
2. Ben-Or, M., Goldreich, O., Micali, S., Rivest, R.L.: A Fair Protocol for Signing Contracts. IEEE Trans. Inf. Theory **36**(1), 40–46 (1990)
3. Asokan, N., Shoup, V., Waidner, M.: Optimistic fair exchange of digital signatures. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 591–606. Springer, Heidelberg (1998)
4. Ateniese, G.: Efficient verifiable encryption (and Fair Exchange) of digital signature. In: Proc. ACM Conf. CCS 1999, pp. 138–146. ACM Press (1999)
5. Bao, F., Deng, R.H., Mao, W.: Efficient and practical fair exchange protocols with off-line TTP. In: Proc. IEEE Symp. Security and Privacy, pp. 77–85 (1998)

6. Kim, T., Huang, L.-S., Perrig, A., Jackson, C., Gligor, V.: Accountable key infrastructure (AKI): a proposal for a public-key validation infrastructure. In: Proc. of the International World Wide Web Conference (WWW) (2013)
7. Haber, S., Stornetta, W.S.: How to Time-stamp a Digital Document. Journal of Cryptology **3**(2), 99–111 (1991)
8. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System. http://bitcoin.org/bitcoin.pdf (2008)