Crossroads
The ACM Student Magazine

Signing Electronic Contracts

by ***David Molnar***

Abstract

Electronic commerce faces the problem of signing electronic contracts. Three approaches for handling electronic contracts include 1) no trusted third party protocols, 2) strongly-trusted third party protocols and 3) weakly-trusted third party protocols. A secondary problem facing electronic commerce is self-enforcing contract design.

Introduction

Contracts are vital for commerce. Every agreement between two or more people involves a contract. Simple things, such as expecting a friend to show up to class, are examples of implied contracts. Other contracts are more formally laid out and signed by all parties involved. After formal contracts, every party receives a copy of the contract. Following the contractual agreement, each party can act with the assurance that other parties will do their share.

Written contracts create mechanisms which allow parties to overcome distrust of each other, because they know misbehavior will be found out. Once written down, the terms of the agreement cannot be forgotten. In cases of cheating or dispute, the "contract" actually refers to that piece of paper which everyone signed. The paper is a convincing certificate to an impartial observer, such as a judge, that an agreement actually occurred.

As more commerce moves online, contracts are moving online as well. Every time anyone orders something from a merchant online, a contract is created promising to exchange money for some good or service. These electronic contracts make electronic commerce possible.

Electronic contracts pose special difficulties. Offline, many contracts are signed face-to-face. All parties signing the contract can be reasonably sure that the other parties are who they claim to be, that the wording of the contract has indeed been agreed upon by all parties, and, eventually, that everyone has signed the contract. None of these are true by default online.

This is perhaps the most serious problem: how do the contracting parties figure out that everyone has signed the *same* contract? The obvious approach would be to do the exact same thing we do offline. This is the focus of the next section.

The "Obvious" Way Of Signing A Contract Fails Miserably

Here is a way to do contract signing online. It is an "obvious" way of doing things because it mirrors how two people in the physical world would sign a contract.

Suppose that two parties, Alice and Bob, are signing a contract. First, Alice and Bob jointly write down the contract, or have their lawyers do it for them. Now both Alice and Bob have a copy of the contract. The contract is not signed yet. Next, Alice signs her copy of the contract and sends it to Bob.

When he receives the signed contract, Bob signs his contract and sends it to Alice. Alice checks to make sure that the contract Bob signed really is the contract she wants him to sign. Bob checks to make sure Alice signed the right contract. Alice signs the contract Bob signed, and sends that to Bob. Bob signs the contract Alice signed, and sends that to Alice.

Now both Alice and Bob have a contract which has both their signatures on it. If Alice wants a return receipt from Bob, then the process is very similar: Alice indicates in her message that she wants a return receipt, so Bob obligingly fills one out and sends it back.

This protocol does not work. Here is a story which shows why:

One day Alice and Bob wanted to sign a contract. "Hey," Alice e-mailed Bob, "I will bet you US$15 that the Nikkei rises 3% tomorrow!" Bob idled a moment, then sent back the message "OK, but if not, then I get your CD collection." After thinking a moment, they decided to make up a real contract ... just in case one of them got "cold feet."

Alice and Bob drew up a contract. Alice signed her copy and sent it to Bob. Then she waited. She waited some more. Still no response from Bob.

Suppose that the next day, the Nikkei rose 2.5%. Eagerly anticipating her money, Alice e-mails Bob, including a copy of their contract. Much to her dismay, Bob's response is "Oh, so that's where that contract went...I thought you didn't want to do that anymore. I mean, I didn't hear anything from you!"

Alternatively, maybe the Nikkei rose 3.5%. The next day, Alice receives an email: "Coming for CD collection. See you at 2:00 - Bob." Alice sits back in her chair and wonders just what happened.

What Went Wrong?

When Bob stopped responding, Alice assumed that he had not received her copy of the contract. In fact, Bob was waiting to see whether he liked the way the contract would turn out. Only after he was sure of the outcome did he decide whether or not to honor the contract. He could do this because Alice was bound to the contract before Bob.

Notice that Alice has no way to determine if Bob's failure to respond is malicious. Furthermore, she has no means of convincing others of his malicious actions. This is the **contract signing problem**.

The Contract Signing Problem

The lack of **atomicity** is the primary cause of the contract signing problem. To say that a series of operations is atomic means that either all of them happen, or none of them happen. A more thorough explanation of how atomicity is important for electronic commerce is given by Tygar [**11**]. When two parties are connected to each other via a possibly unreliable network, ensuring atomicity becomes a serious problem.

In a **contract signing problem**, there are two or more parties who are trying to agree on a contract. Each party will digitally sign the contract to signal their agreement.

A variant of the contract signing problem is the **return receipt problem** or **certified mail problem**. Here, Alice is sending mail to Bob, and she wants some evidence that Bob received the mail. This is crucial for such applications as subpoena or mail-order business. A return receipt protocol tries to ensure that Bob gets the mail only if Alice

gets a receipt which will convince everyone that Bob actually got the mail.

Both these problems can be thought of as special cases of the **fair exchange problem**: Alice and Bob each have something the other wants. They want to exchange their items such that Alice gets Bob's item if and only if Bob gets Alice's item. In the case of contracts, the item is a signature on a shared contract. All of these have the same problem with respect to the lack of atomicity already noted.

How To Fix It?

At this point, note that the problem is *independent* of what cryptographic tools we used to sign the contract. It is independent of whatever kind of money used for fulfilling the contract. It does not matter if the signature is totally unforgeable, or if the money is counterfeit-proof. The exact details of how digital signatures are done are irrelevant. Cryptography alone will not solve the contract problem.

Instead, the right approach is to use cryptographic tools to build **protocols**: explicitly specified processes for solving problems.

This problem is not new to electronic commerce. The same problem of atomicity exists with contracts over fax machines or telephones. In these cases, however, often the two parties contracting know each other and have done business before. If Bob's fax machine suddenly "breaks," Alice can call Bob and see what has happened. If Bob refuses to answer, Alice becomes suspicious. Because the two parties know each other, they have a wider range of actions available to them.

In U.S. law, the problem is recognized by the Uniform Commercial Code [**12**]. Contracts in excess of $500 carried out over wires require a separate written contract. Business-to-business electronic contracts are handled by signing such a contract which specifies a kind of electronic data interchange system and provisions in case the system fails. Not only does this make the contract legally binding, but it also gives the parties assurance that everything will work as expected.

This situation does not hold in general. Alice and Bob may not know each other. In particular, consider the case of a merchant who wants to cater to many different customers online. It is impractical for the merchant to form a separate written contract with each of these customers. At the same time, both merchant and customer need some assurance that they can create some kind of contract. This is where protocols for

contract signing are helpful.

Requirements for Contracts

Here is an outline of some requirements for a two-party contract signing protocol:

- **Fairness:** Neither party should be bound to the contract without the other party being bound as well. If one party decides to abort the contract signing protocol, the other party should know that the protocol is over.
- **Completeness:** The protocol should be robust against adversaries attempting to cause it to abort without the consent of either party.
- **Non-Repudiation:** The protocol should not allow parties to arbitrarily decide to withdraw their support from a contract after the protocol is over. If Alice and Bob sign a contract, Alice should not be able to break the contract without Bob's consent and vice versa.
- **Efficiency:** Signing the contract should require a minimum of messages and computation time.

Two-Party Only Protocols

Now consider protocols which involve only Alice and Bob. There is a pessimistic result: according to some assumptions, we can prove that the problem is "impossible." Happily, these assumptions do not cover all possible protocols. Some two-party alternatives do exist.

The upside of these alternatives is that they can be used even in situations where Alice and Bob do not trust each other, or anyone else, **at all**. The downside is that because of this restrictive requirement, these protocols are inefficient.

Deterministic Two-Party Protocols Are Impossible

Start out by trying to build a deterministic protocol for solving the fair exchange problem. Anything that solves fair exchange can solve contract signing -- here what each party has that the other wants is the signature on a contract. (This is the reason that contract signing protocols are sometimes referred to as "fair exchange of digital signatures" in some papers).

Here is the problem: at some step, either Alice or Bob has sent part of the message. So either Alice or Bob has an advantage, in terms of having more of the contract.

Whichever party has an advantage can, at that point, simply decide to stop participating in the protocol. This decision is silent and undetectable by the other party. For a deterministic two-party protocol without cryptography, one party always has an advantage, and aborts cannot be detected. Therefore these protocols cannot provide satisfactory contract signing.

Ralph Merkle's Puzzle Protocol

An ingenious alternative makes use of a cryptographic primitive called a **puzzle**. First introduced by Ralph Merkle [**4**], a puzzle is a string which takes a precisely known amount of time to decrypt. For example, one way to create a puzzle is by encrypting a message with a symmetric cipher and a very short key of 20 bits. Assuming that no better attack exists on the cipher than brute-force, anyone given the puzzle will have to try every possible 20-bit key before decrypting the puzzle. It will take 2^20 operations to search the entire keyspace, and we expect to find the right key halfway through, so the puzzle will take about 2^19 operations. Given knowledge of how long an operation takes, this translates into some amount of time.

Puzzles can be used to solve the abort problem. Alice receives a return receipt from Bob as follows:

1. Alice creates a puzzle from a signed contract. The time to decrypt for this puzzle is very high, so high that it is impractical for Bob to wait this long to read the contract. One way she may do this is by encrypting her signed contract with a very long, randomly chosen key in some symmetric algorithm like the Data Encryption Standard.
2. Alice sends the puzzle to Bob. Alice asks Bob to send her a return receipt for the puzzle.
3. If Alice receives a return receipt, she sends Bob a "hint" for the puzzle. This hint allows Bob to solve the puzzle much faster than computing it himself, but does not reveal the entire message. For example, if the puzzle took the form of a message encrypted with a long key, Alice sends Bob the first few bits of the key. Alice asks for a return receipt for this "hint." No return receipt means no more hints.
4. The protocol continues until Alice has received a return receipt for enough "hints" such that solving the puzzle becomes trivial. Now she knows that Bob has the message.

Because Bob sent Alice a return receipt for the puzzle, Alice knows that Bob has enough information to reconstruct the message. The only question is how long it will take. Since the puzzle starts out being unreasonably difficult to solve, Bob needs Alice's hints to get the message. Alice gives no new hints until Bob sends a return receipt for the past hints, and so she knows, up to the last receipt, how much of a hint Bob has. By obtaining a receipt for each hint given to Bob, Alice always knows how much time is left, give or take a hint, until Bob can read the message. After that time, she is justified in acting as though Bob had the message.

There are a few subtleties with this protocol. The first is that the "puzzle" described above actually is not sufficient. The problem is that searching for a key is extremely easy to speed up by distributing among many computers (see **http://www.distributed.net/** for a real world example). This makes it difficult to know how long such a puzzle will really take to decrypt, unless Alice knows for sure how many friends Bob has.

Rivest, Shamir, and Wagner have given explicit constructions for puzzles based on modular exponentiation which are conjectured to be "inherently sequential" [**8**]. An inherently sequential process is one for which multiple machines can not speed up the process by a large amount; the only way to solve the puzzle faster is to find a single bigger machine. This defeats distributed attacks, and so allows the time bound to be made precise, given knowledge of the machine attacking the puzzle.

A second subtlety concerns the possibility of obtaining partial information from intermediate stages of the puzzle. Can we guarantee that the "hints" Alice sends to Bob reveal no partial information about the plaintext? We can fix this by applying an "all or nothing transform", or **AONT**, to the message before making it into a puzzle. First proposed by Rivest [**7**], an AONT ensures that no partial information is available from the message until the entire message is reconstructed. Recently, Boyko produced an AONT based on the popular "optimal asymmetric encryption padding" scheme [**2**]; this AONT allows us to prove that no partial information leaks from the puzzle until it is entirely reconstructed.

This is part of a class of **gradual exchange** protocols. The aim of a gradual exchange is to make the advantage possessed by any one party uncertain enough so as to make continuing with the protocol that party's best option. Another example of such protocol is found under the "Digital Certified Mail" heading in Applied Cryptography [**9**]; many

other examples exist in the literature.

The major drawback of all such gradual exchange protocols is that their gradual nature necessitates extremely large numbers of messages. For many applications, the overhead is prohibitive.

Trusted Third Party Protocols

In the "Real World," return receipts and contracts at a distance usually involve trusting someone else. For example, the Post Office is trusted to extract return receipts from people before giving them registered mail. This works because the Post Office is trusted to collect receipts and not to forge receipts for packages which were not sent. In this sense, the Post Office is a **trusted third party.** Sometimes, as in Bruce Schneier's *Applied Cryptography*, this trusted third party is called *Trent*.

Carrying this model over directly to the online world means that every time she wants to make a contract or obtain a return receipt, Alice sends her message to Trent. Then Trent sends it on to Bob. Now if Bob does not respond, Alice can say she sent the message to Trent. Trent will back her up, and everyone will believe the two of them because that third party is trusted.

The problem here is that the third party needs to be involved in every transaction. Worse, an evil third party can fabricate contracts between Alice and Bob. This means Alice needs a high-bandwidth third party with very, very strong trust requirements. The first problem can be addressed by adopting a so-called "optimistic" approach, and the second by using a third party which does not have the power to create contracts.

Optimistic Third Party Contract Signing

The contract signing protocol just described uses an online third party for each and every transaction. Recently, some researchers have considered the so-called "optimistic" approach: instead of involving the third party in every transaction, assume that most transactions are going to execute correctly. Then invoke the third party only for those (hopefully rare) cases in which something goes wrong.

Micali's Certified E-Mail With Invisible Post Offices

One of the first optimistic protocols is due to Silvio Micali and was presented at the

1997 RSA Data Security Conference [5]. Micali also patented the technique. The protocol provides return receipts - Alice sends Bob a message and knows Bob can read it if and only if Bob sends Alice a receipt. The protocol looks roughly like this:

1. Alice sends Bob the message encrypted with the public key of the trusted third party. Now Bob only has an encrypted message.
2. Bob sends Alice a receipt for the encrypted message.
3. Alice sends Bob the unencrypted message. Now Bob can encrypt it with the public key of the trusted third party and be see if it matches what Alice gave him in step 1.

If there is ever a problem, Alice or Bob run an **abort protocol**: an appeal to the trusted third party to either stop protocol or resolve any difficulties. Since the trusted third party knows its own private key, it can decrypt the message sent to Bob in step 1, even without Alice's help. This allows Bob some protection if he sends Alice a receipt but never receives the message.

Asokan, Shoup, and Waidner: Fair Exchange of Digital Signatures

At EUROCRYPT '98, Asokan, Shoup, and Waidner extended optimistic protocols to fair exchange [1]. In their protocol, Alice and Bob send encrypted signatures to each other. Then they prove to each other that the encryptions really do contain digital signatures, and that these are signatures on the correct documents. Finally, they send each other decryption keys.

If something goes wrong, a third party can step in and use the "proofs of correct encryption" to verify that all has gone well up to a point. Then the third party can take some course of action, such as terminating the protocol or decrypting the signatures for both parties.

A New Requirement: Jakobsson, Garay, and MacKenzie's Abuse-Free Optimistic Contract Signing

At CRYPTO '99, Jakobsson, Garay, and MacKenzie [3] introduced a new requirement for optimistic contract signing protocols:

- **Abuse-freeness:** Neither party signing a contract should be able to prove to

any other party what the terms of the contract are, until after the contract is signed. In fact, they should not be able to prove to anyone else that they are engaged in signing a contract at all.

This requirement is useful in such situations as job offers. Currently, if Alice offers Bob a job for some amount of money, Bob can turn around and show the offer to Carol. Carol may then decide to raise her offer to Bob, which Bob can then show to Alice. It may be the case that Alice would prefer this did not happen.

Note that the final contract is still verifiable by anyone; it must be in order to be of use. The "abuse-free" requirement simply says that the contract negotiations must be kept between the contracting parties.

This requirement is implemented by the use of a primitive called **Designated Verifier Proofs** . A designated verifier proof is a statement of the form "Either X is true, or I know V's secret key." The party V is called the "designated verifier."

Now in the case that "X" means "Alice has signed this contract," Alice can make up the statement "Either Alice has signed this contract or I know Bob's secret key." Alice can prove this statement to Bob, since she can sign the contract. If Bob tries to show the statement to someone else, however, then he has a problem. He could have made up the statement himself, because it is also true if "I know Bob's secret key" -- and Bob hopefully knows his own secret key.

These designated verifier proofs are then combined with the above idea to produce a fair optimistic contract signing protocol.

Breaks in Optimistic Protocols

Optimistic third party protocols seem like a great idea. Unfortunately, two of the optimistic contract signing protocols given to date were flawed in their initial presentation. These protocols have since been fixed, but the attacks reveal that these protocols can harbor very subtle flaws.

For the Asokan, Shoup, and Waidner protocol, Paul Syverson pointed out a flaw in the rump session of CRYPTO '99: by suitably manipulating the appeal to the trusted third party, Alice can sometimes cheat Bob. She does this by waiting until after Bob has sent her a message, and then telling the trusted third party that the protocol should abort.

The TTP causes an abort, leaving Alice with a receipt and Bob thinking that the protocol failed.

In the case of abuse-free contract signing, Mitchell and Shmatikov applied formal protocol verification techniques to the protocol introduced at CRYPTO '99 [10]. They had been trying to prove that the protocol was both fair and abuse-free. This failed, because the protocol as stated has a subtle flaw which allows for a third party to be convinced of the contract's existence in some special cases. Fortunately, the authors managed to find a small variant which fixes this flaw -- but the experience demonstrates the difficulty of designing and verifying such protocols.

Not-So-Trusted Third Party Protocols

These are protocols which require third parties, but do not have to trust the third parties very much. In particular, the third parties are unlikely or unable to forge contracts.

The Power of A Bulletin Board For Return Receipts

Riordan and Schneier proposed a protocol for return receipts which uses a "public bulletin board" [9]. This is a central message server that everyone can look at, anyone can add to, but does not allow anyone to erase messages.

For the case of Alice sending a message to Bob, the protocol looks like this:

1. Alice chooses a random symmetric key and encrypts a message M with this key. She sends this encrypted message to Bob. She does not send Bob the key.
2. Bob sends to Alice this signed message: **I would like Alice to post the key for this encrypted message M on the bulletin board by time T.**
3. Alice posts the key on the bulletin board before time T.
4. Bob downloads the key and decrypts M.

Now if something goes wrong, Alice can produce Bob's signed message asking her to post a key for a particular message. Then the bulletin board can be consulted to see if the key really was posted. Finally, the key can be checked to see that it really does decrypt the message.

This allows Alice to be confident that Bob will not receive the message without sending

her some kind of return receipt first.

Rabin's Random Beacons

In 1981, Michael O. Rabin suggested a protocol based on **random beacons** [6]: machines whose only purpose in life is to pick a random number between 1 and n at every time interval *t* and then broadcast it to the world. Alice and Bob both have access to the output of the random beacon. The protocol works like this:

1. Alice and Bob make the following agreement: *At each interval t, we will each pick a random number and broadcast it to the world before the beacon does. If it should ever happen that we both pick the same number as the random beacon, then we will both be bound to the contract.*
2. Alice picks a random number, time-stamps it, and broadcasts it.
3. Bob picks a random number, time-stamps it, and broadcasts it.
4. The beacon picks a random number, time-stamps it, and broadcasts it.
5. If Alice receives Bob's number and the beacon's number within a certain timeout period, and these numbers are the same, she knows she is bound to the contract. The same follows for Bob. If Alice does not receive Bob's number or the beacon's number within a certain timeout period, she stops playing.

Now neither Alice nor Bob can be bound to the contract unless both are bound to the contract. This is because they must both guess the same number at the same time interval. The random beacon prevents attacks in which Alice or Bob play with their time stamps, or refrain from answering until it is convenient for them.

How To Pick A Protocol

Which of these protocols is best? The answer depends on several factors:

1. Resources Available: You need a trusted third party for a protocol which uses a third party. Where do you find one? The "gradual exchange" protocols considered require a large amount of communication and computational overhead. Can you afford that in your application?
2. Confidence in protocol: are you sure the protocol is correct? Some of these protocols are easier to verify than others. For example, the first run of "optimistic" contract signing protocols had bugs in them. These bugs were caught by the application of protocol verification techniques - but did they catch

everything?

Conclusion: Now That We Have A Protocol, What Do We Do With It?

Finally there are some ways of approaching the contract signing problem. The next step for electronic commerce is to figure out how to specify the contracts we want to sign. Recently, companies such as InterTrust have arisen which provide a kind of "contract language" for parties to write software which monitors and enforces contract obligations for intellectual property. Another effort in this area can be seen at www.erights.org, which has fielded a language called "E" for writing contracts. All of these require secure contract signing as a means for agreeing on these contracts in the first place.

Acknowledgements

The author is grateful to the anonymous reviewers for their valuable comments. Ron Rivest described Micali's contract signing protocol. Markus Jakobsson and David Wagner were kind enough to answer questions about their respective protocols.

References

1

Asokan, N. , Shoup, V, and Waidner, M. Optimistic Fair Exchange of Digital Signatures. In *Proceedings of EUROCRYPT '98*

2

Boyko, V. The Security Properties of OAEP as an All-or-nothing Transform. In *Proceedings of CRYPTO '99* . Springer-Verlag LNCS 1666 1999, see also **http://theory.lcs.mit.edu/~boyko/aont-oaep/paper.ps**

3

Jakobsson, M., MacKenzie, P, and Garay, J. Abuse-Free Optimistic Contract Signing. In *Proceedings of CRYPTO '99* Springer-Verlag LNCS 1666 1999, pp. 406-416.

4

Merkle, R. Secure Communications over insecure channels. Communications of the ACM 21:294-299, April 1978.

5

Micali, S. Certified E-mail with Invisible Post Offices. Lecture at RSA Security Conference 1997.

6

Rabin, M.O. Transaction protection by beacons. Journal of Computer and System Sciences, 27(2):256-267, October 1983

7

Rivest, R. All-or-nothing encryption and the package transform. In Eli Biham, editor, *Fast Software Encryption : 4th International Workshop*. Springer-Verlag LNCS 1267 p.210-218.

8

Rivest, R. Shamir, A. and Wagner, D. Time-lock puzzles and time-release crypto. MIT LCS tech report TR-684, February 1996. **http://www.cs.berkeley.edu/~daw/papers/timelock.ps**

9

Schneier, B. and Riordan, J. A Certified E-Mail Protocol with No Trusted Third Party. 13th Annual Computer Security Applications Conference, ACM Press, December 1998 **http://www.counterpane.com/certified-email.html**

10

V. Shmatikov, J. Mitahell. Analysis of Abuse-Free Contract Signing. Proceedings of Financial Cryptography 2000.

11

Tygar, D. Atomicity in Electronic Commeace. In *Internet Besieged*. Addison-Wesley and ACM Press. October 1997, pages 389-406. (A longer, earlier version appeared in Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing, May 1996, pages. 8-26.) See also **http://www.cs.berkeley.edu/~tygar/recommend.html**

12

The Uniform Commercial Code of the United States **http://www.law.cornell.edu/ucc/ucc.table.html**

Biography

David Molnar began using PGP in 1993. He became interested (obsessed?) with figuring out "why it worked" and has been studying cryptography ever since. Now an undergraduate at Harvard University, he keeps up with security issues via courses, reading newsgroups, mailing lists, and conference papers, and attending DEF CON in his home city of Las Vegas. David is an ACM Student Member and a member of the International Association for Cryptologic Research.