A Fair Protocol for Signing Contracts

(Extended Abstract)

Michael Ben-Or [1] Oded Goldreich [2] Silvio Micali [3] Ronald L. Rivest [4]

ABSTRACT

Assume that two parties, A and B, want to sign a contract over a communication network, i.e. they want to exchange their "commitments" to the contract. We consider a contract signing protocol to be *fair* if, at any stage in its execution, the following hold: the **conditional probability** that party A obtains B's signature to the contract *given* that B has obtained A's signature to the contract, is close to 1. (Symmetrically, when switching the roles of A and B).

Contract signing protocols cannot be fair without relying on a trusted third party. We present a fair, cryptographic protocol for signing contracts that makes use of the *weakest possible form of a trusted third party (judge).* If both A and B are honest, the judge will never be called upon. Otherwise, the judge rules by performing a simple computation, without referring to previous verdicts. Thus, no bookkeeping is required from the judge. Our protocol is fair even if A and B have very different computing powers. Its fairness is proved under the very general cryptographic assumption that functions that are one-way in a weak sense exist. Our protocol is also optimal with respect to the number of messages exchanged.

---

[1] Institute of Mathematics and Computer Science, Hebrew University, Jerusalem, Israel. Work done when visiting MIT's Lab. for Comp. Sc. Supported by a Weizmann Postdoctoral Fellowship.

[2] Computer Science Dept., Technion, Haifa, Israel. Currently in the Lab. for Comp. Sc., MIT. Supported by a Weizmann Postdoctoral Fellowship.

[3] Laboratory for Computer Science, MIT, Cambridge, MA 02139, USA. Supported by NSF Grant DCR-8413577 and an IBM Faculty Development Award.

[4] Laboratory for Computer Science, MIT, Cambridge, MA 02139, USA. Supported by NSF Grant MCS80-06938.

1. Introduction

Let A, B,... be users who can exchange messages over a communication network. For example, they may be the users of the ordinary mail system, or of a telephone network, or a modern computer network.

We will assume that a *Signature Scheme* S is adopted in the network. Two key properties are required from a signature scheme. First, *unforgeability*: only user U can create U's signature on message m. Second, *universal verification*: any other user should be able to verify that U's signature on message m is indeed a valid one. An instance of a signature scheme may be provided by ordinary "hand-written" signatures. Hand-written signatures are believed to be hard to forge and can be universally verified as trusted notary publics keep samples of each user's signature. Another instance of a signature scheme, more suitable for computer networks, is a "digital signature scheme". This notion was introduced by Diffie and Hellman [DH] and first implemented by [RSA]. The strongest notion of security for a digital signature scheme was suggested and concretely implemented (based on a weak and general complexity assumption) by Goldwasser, Micali and Rivest [GMR]. An historical account on the problem of digital signature can be found in [GMR].

We now describe the problem of "fair" contract signing. Two users, A and B, have negotiated over the network a contract *CONT* and now want to obtain each other's signature on it. In essence, the problem of signing a contract consists of exchanging signatures of an ordinary message, but with the additional constraint that the exchange must be "simultaneous". Therefore,

, *"a signature to a contract" does not necessarily consist of of applying the signature scheme to the text of the contract.*

In general, a signature of A to *CONT* is a set of messages that only A can generate on input *CONT*, e.g. *some* of these messages may be generated by applying the signature scheme S. A good contract signing protocol should satisfy the following (informal) conditions.

1) *Viability*: If both parties follow the protocol properly, then at its termination, each will have his counterpart's signature to *CONT*.

2) *Fairness*: If one party, say A, follows the protocol properly then at any stage during its execution, B has A's signature on *CONT* if and only if also A has B's signature on *CONT*.

The fairness condition is hard to satisfy, essentially because "simultaneity" is hard to meet in our discrete world [EY]. Thus, in order to successfully implement a contract signing protocol, it is necessary to relax and, at the same time, to formalize the fairness condition. The meaningfulness of a solution to the contract signing problem will depend on the acceptability of the definition used to approximate the intuitive notion of fairness. Two main approaches to approximating fairness have been considered. The first one, interprets simultaneity as deriving from equal computational effort. This approach, meaningful only if both parties are assumed to have equal computing power, suffers from some inherent additional disadvantages. The second approach, interprets the simultaneity of two events as very high probability that one event happens if and only if the second event happens.

1.1 Computational Approaches to Fairness

Even [E], Blum [B] and Even, Goldreich and Lempel [EGL], proposed a computational interpretation of approximate fairness. This approach requires that at any stage during the execution of the protocol, the computational effort required from the parties in order to get each other's signature to *CONT* be approximately equal. Informally, a protocol is said to satisfy this definition of fairness if, during its execution, for both parties the computational difficulty of computing the counterpart's signature to CONT decreases by equal amounts at each step, till it vanishes.

This approach suffers from the following major weaknesses.

1) This definition of fairness is meaningful only if both parties are assumed to have "equal computing power". Otherwise, at some step of the protocol, one party may find the remaing computational effort, necessary to obtain his counterpart's signature, to be feasible; while the same effort may be beyond the power of his counterpart. Such "equal computing power" was assumed by all the above mentioned researchers.

   We feel that assuming "equal computing power" is both unrealistic in practice and undesirable from a theoretical point of view. In real life, parties may often have different computing power (e.g. consider a large commercial firm and an individual). Also, it is difficult for a party to estimate accurately the computing ability of the other party (one can always pretend to be less powerful).
   [Jumping ahead, let us point out that the probabilistic approach, as well as our protocol, is valid even if the parties have very different computing power.]

2) As observed by Rackoff, this approach to fairness is *prone to the devastating effect of "early stopping"*. Assume party B stops the execution of the protocol prematurely, at a point when each party will need 10 years of computing time to obtain his counterpart's signature on $CONT$. Should A keep computing for the next 10 years or should she give up and hope that B is doing the same and thus will never have her signature? The situation effectively binds A to the contract without offering any priviledges. In fact, if she acts as if the contract is not binding, B has the option, by investing 10 years of computation time, to enforce the contract and to put her in serious legal trouble. Though this approach was developed for two parties only, it should be noticed that *the dilemma created by early stopping cannot be settled in court by a judge that rules deterministically.*
   [If the judge is allowed to rule probabilistically, then our protocol is optimal. In particular, if one party stops prematurely, the other will invoke the (probabilistic) judge with the following guarantee. If the judge rules that $CONT$ is binding when B appeals then, with high probability, he also rules so in case A appeals.]

3) A third difficulty arises with this approach that may not be inherent, but is certainly a formidable one: the difficulty of proving the fairness of a protocol in this sense. Proving that certain problems cannot be efficiently solvable seems to be hard. Proving that the computational difficulty of certain problems decreases by a fixed amount, when releasing some "partial information", seems even harder. The correctness of Blum's protocol [B] follows from the assumption that a particular computational problem, related (but not known to be computationally equivalent) to integer factorization, is infeasible. Unfortunately, this problem has been shown to be efficiently solvable by Hastad and Shamir [HS]. The correctness of Even Goldreich and Lempel's protocol [EGL] follows from the assumption that "ideal" trapdoor one-way permutations exist and the assumption that "uniformly-hard" one-way functions f exist. By this they mean that given $f(x)$ and k bits of (information about) x, where x is n-bit long, the best algorithm for computing x essentially consists of trying all the 2^{n-k} possibilities for the remaining bits. This is a very strong assumption. Even's protocol [E] (as well as its simplified version [G]) requires, in addition to the existence of a uniformly-hard one-way function, an extra non-mathematical assumption: that the contract's subject has a fixed and known value.
   [In comparison, the correctness of our protocol follows from a much weaker and more general complexity assumption: the existence of one-way functions f. Such f's need not be "ideal" or uniformly-hard.

In particular, it may be that half of the bits of x are easy to compute on input $f(x)$. We only require that (all of) x is not efficiently computable on input $f(x)$. In case the communication network is a computer network or a telephone network, we also need particularly strong digital signature schemes. Such schemes have been proved to exist under simple complexity assumption by [GMR].]

1.2 A Probabilistic Approach to Fairness

Rabin [R] proposed to sign contracts with the help of a trusted third party that, at regular intervals (say each day), publicly broadcasts a randomly chosen integer between 1 and 100. Parties A and B can then sign contracts by agreeing on a future date D and exchanging signed messages of the form: "I am committed to *CONT* if integer i is chosen at date D". A goes first. B will send his i-th message *only if* he gets A's i-th message Similarly A sends her $i+1$-st message *only if* she receives B's i-th message. It should be stressed that all messages should be exchanged prior to date D. B may try to cheat by not sending his i-th message after receiving A's i-th message. The advantage he gains by doing so is not too large: the probability that he will have a signed contract, on date D, but A will not, equals 1/100. In this sense, the protocol is "fair".

How can this be made formal? We have two possible alternatives:

1) Requiring that the probability that "B has A's signature on *CONT* but A does not have B's" is small.

2) Requiring that the difference between 1 and the conditional probability that "A has B's signature on *CONT*", given that "B has A's signature to *CONT*", is small.

Notice that the second condition implies the first. We believe that the conditional probability is a better measure for fairness, since it better models the intuitive notion of fairness as simultaneity ("if B has the signature then A has it too" and vice versa).

In formalizing the definition of fairness we will use a "security" parameter k which will constitute, together with *CONT*, the input to the protocol. We also use a function μ from integers to the 0-1 inteval. μ will serve as our measure of "negligible" probability; namely, we will not care about events occuring with probability less than $\mu(k)$. For example, one may choose $\mu(k)=2^{-k}$ or $\mu(k)=k^{-c}$ for some positive integer c. The parties are allowed to make random choices during the execution of the protocol. So is the third party, if he is called into play.

Definition: A contract signing protocol is *μ-fair for A* if, on input k and *CONT*, in case A follows the protocol properly, the following holds. At any stage, during the execution of the protocol, in which the probability that B has A's signature to *CONT* is greater than $\mu(k)$, the **conditional probability** that "A has B's signature to *CONT*", given that "B has A's signature to *CONT*", is at least $1 - 1/k$.
Here the probabilities are taken over all the random choices of A, B and the third party (in case he is called into play).
A protocool is μ-fair if it is μ-fair both for A and B.

For example, if the trusted third party of Rabin's protocol is parametrized, so that on input k it randomly selects an integer between 1 and $l(k)$, then the protocol would be μ-fair if and only if $l(k)\geq\mu(k)^{-1}\cdot k$. In fact, after A has sent i messages, but before B has replied, the conditional probability that "A has B's signature to the contract" given that "B has A's signature to the contract" equals $1-1/i$.

1.3 Intervention by a Third Party

We believe that the definition of μ-fairness is the correct one. However, it cannot be enforced without the intervention of a third party.

Theorem 1: Let $\mu(k)$ be smaller than 1, for every k. Then no viable, μ-fair two-party protocol for signing contracts exists, without the intervention of trusted third parties.

A proof of Theorem 1 will appear in the final version of this paper. The proof is not hard once the right formalization is reached.

Relying on the existence of a trusted third party is indeed a drawback, but preferable to assuming equal computing power. It is certainly preferable to accepting the disruptive effect of "early stopping" inherent in the computational approach to fairness.

Since a μ-fair protocol must rely on a trusted third party, its quality will depend on the role that such a party plays in it: The more "inconspicuous" and efficient the third party is, the better the solution is. The third party we use is a probabilistic judge (algorithm). The judge is inactive until he is invoked. The judge can rule on whether a contract is binding, in the presence of only one party. Furthermore, the judge is invoked only in case of dispute and does not need to store past verdicts. The mildness of the (third party's) intervention in our solution can be demonstrated by comparing it with other forms of trusted third parties proposed in previous solutions to the contract signing problem.

A simple, folklore solution to signing contracts is a cancellation center which stores all invalidated contracts (see [EGL]). A contract is signed by exchanging messages of the form: "I'm committed to *CONT* unless I've deposited a cancellation notice in the center by date D", where D is some future date. This naive solution suffers from serious practical drawbacks. The paperwork involved in maintaining the cancellation center is tremendous. Even more disturbing is the fact that if A wants to convince C that she (A) has B's signature to *CONT*, A must get a certificate from the cancellation center that *CONT* was not cancelled. This is the case even if both A and B are honest and wish to execute the protocol properly.

As discussed in section 1.2, Rabin [R] proposed the use of a trusted third partywhich broadcasts randomly chosen integers at fixed times. We point out that this third party must always be active, regardless of the honesty of all parties and of whether its output is being referred to.

2. Optimal Fair Protocols for Parties with Different Computing Power

In this section we develop a viable and fair protocol, without relying on the equal resources assumption. During the execution of the protocol, the probability that a party obtains his counterpart's signature to *CONT* grows from zero to one. This growth must be moderate to satisfy the fairness requirement.

In subsection 2.1, we give a tight lower bound on the number of messages exchanged in a viable, μ-fair protocol. In subsection 2.2, we exhibit a protocol which achieves this lower bound. The heart of our solution is the efficiency of the implementaion of the judging procedure, suggested in subsection 2.3.

2.1 On the Number of Messages Exchanged in a μ-Fair Protocol

A protocol for contract signing is a sequence of message exchanges hereafter called *steps*. In each step one of the parties sends a message to the other. Without loss of generality, no party sends messages in two consecutive steps; i.e. each party alternately sends and receives messages. Let us denote the party which sends [receives] a message in step i by S_i [R_i]. Let E_i denote the event "after step i, party R_i has S_i's commitment to *CONT*", i.e. enough information so that the judge will rule that the contract is binding for S_i.

Following is an analysis of the implication of the μ-fairness requirement on the number of steps in a viable protocol. This number may be a function of the security parameter k. For simplicity, we assume here that the number of steps is independent of the contract *CONT*. Let us denote by $\#(k)$ the number of steps in a viable μ-fair protocol, on input the security parameter k and the contract *CONT*. By the viability requirement, we have

$$Prob(E_{\#(k)}) = 1$$

(since upon termination of the protocol the probability that each party has his counterpart's signature to *CONT* is 1.) By the μ-fairness, if $Prob(E_i) \geq \mu(k)$ $(1 \leq i \leq \#(k))$ then

$$Prob(E_{i-1}|E_i) \geq 1 - \frac{1}{k}.$$

This implies

$$Prob(E_{i-1}) \geq (1 - \frac{1}{k}) \cdot Prob(E_i)$$

(since $\frac{Prob(A)}{Prob(B)} \geq Prob(A|B)$ for any two events A and B).
In order to minimize $\#(k)$ we set $Prob(E_i) = (1 - 1/k)^{\#(k)-i}$, $1 \leq i \leq \#(k)$. Finally, $Prob(E_1)$ is set to $\mu(k)$, so that the fairness requirement is not violated by the first step. Thus, $\#(k)$ and the $Pr(E_i)$'s are easily bounded by expressions depending on k and $\mu(k)$. Namely,

Theorem 2: Every viable, μ-fair protocol for signing contracts has length at least $\#(k) = k \cdot \log \mu^{-1}(k)$. Furthermore, the probability that after step i one party has his counterpart's signature to *CONT* does not exceed $(1 - 1/k)^{\#(k)-i}$. (The logarithm is taken to the natural base.)

A reasonable choice of $\mu(k) = poly(log\ k)^{-1}$ yields $\#(k) = \Omega(k \cdot \log k)$.

2.2 The Proposed Protocol

Our protocol makes use of the signature scheme of the network. For the purpose of this extended abstract, it will be convenient to decouple the analysis of a contract signing protocol from the security of the underlying signature scheme used in its implementation. In fact, if the network is, say, the ordinary mail system, the signature scheme in use (like pen-written signatures, fingerprints, etc.) may not be easy to analyze mathematically. This decoupling can be done by assuming that the signatures used in the implementation are totally unforgeable. Namely, for each message m and user A, the probability that another user (B) can produce A's signature on m is zero. (Independent of B's computing power.) It should be stressed that *digital* signature schemes cannot possibly be unforgeable in this sense, and problems might arise when implementing our protocol with a "concrete" digital signature scheme. In fact, the "security" of a contract signing protocol cannot exceed the "security" of the underlying signature scheme, but it may be *much* less. This is so because the protocol, in its concrete implementation, may interact badly (in an unpredictable manner) with the signature scheme. However, this is not the case with respect to the protocol presented in this extended abstract:

Our protocol remains fair when implemented with high quality signature schemes as the ones in [GMR].

The formal version of the above statement and its proof will appear in the final write-up. We now present our protocol for signing contracts. As before, the parties to the protocol will be denoted by A and B and the contract they wish to sign will be denoted by *CONT*. We assume that *CONT* also specifies the security parameter k and the function μ, the name of the party that goes first in the protocol (A in the description below) and the name of the party which goes second (B). Recall that $\#(k) = k \cdot \log \mu^{-1}(k)$.

The Protocol for A and B

0A) A chooses $l=\lceil \#(k)/2 \rceil$ arbitrary messages $a_1, a_2, ..., a_l$, with the only restriction that they were not used before by A.
(A's signatures to these messages will be hereafter referred to as A's secrets.)
A sends to B the following declaration (signed): "$CONT$, $a_1, a_2, ..., a_l$".

0B) B chooses l arbitrary messages $b_1, b_2, ..., b_l$, with the only restriction that they were not used before by A.
(B's signatures to these messages will be hereafter referred to as B's secrets.)
B sends to A the following declaration (signed): "$CONT$, $b_1, b_2, ..., b_l$".

(for $1 \leq i \leq l$)

iA) A sends her signature on a_i to B.

iB) B sends his signature on b_i to A.

Handling early stopping : If the entire protocol is not completed within T time units, party X invokes the judge with inputs the initial declarations (steps 0A and 0B), $CONT$, X, and the secrets of Y (X's counterpart) in X's possession.

The Judge's Procedure

On inputs the signed initial declarations, $CONT$, X and m secrets of Y, the judge checks whether the secrets are valid signatures, of X's counterpart, according to the protocol. If so, he then chooses an integer e (hereafter referred to as *the edge*) between 1 and $2l$ *as specified in section 2.3.* The same edge e is chosen every time $CONT$ is brought before the judge. The probability distribution of the edge is, essentially, the following.

$$e = 1 \text{ with probability } \quad p_1 = (1-\frac{1}{k})^{2l-1}.$$

$$e = i,\ 1 < i \leq 2l, \text{ with probability } \quad p_i = (1-\frac{1}{k})^{2l-i} - (1-\frac{1}{k})^{2l-(i-1)}.$$

If $X=A$ and $m \geq \lfloor e/2 \rfloor$ or if $X=B$ and $m \geq \lceil e/2 \rceil$ then the judge rules that $CONT$ is binding.
In this case, the judge waits for T' time units to pass and then sends his signed verdict to both parties.

Notice that if both parties are honest and complete the protocol within T time units, the judge will never be invoked. In fact both will have all the counterpart's secrets, a number greater than or equal to any possible choice of the edge. This fact can be easily verified by any other party. Problems may arise only if one of the parties prematurely terminates the protocol, say after t ($<\#(k)=2l$) steps. In such a case determining the "sufficient number of counterpart's secrets" is of importance and is done by the judge as described above. Notice that the parties cannot know the value of the edge during the execution of the protocol. This is achieved by delying the answer of the judge. Note that the above probability distribution by which the edge is chosen makes the protocol μ-fair and corresponds to the probability distribution of Theorem 2. Thus our protocol is of shortest length among all the μ-fair protocols. Namely,

Theorem 3: For function μ, there exists a viable, μ-fair protocol, for signing contracts, of length $\#(k) = k \cdot \log \mu^{-1}(k)$. (Here the logarithm is taken to the natural base.)

2.3 On the Determination of the Edge

Selecting the edge so that $Prob(e=i)=p_i$ is easy of one flips unbiased coins. (Let $q_0=0$, $q_i=\sum_{j=1}^{i} p_j$. Randomly select 0 and 1 with equal probability to construct the binary expansion a real number α between 0 and 1. Stop the construction of α as soon as $q_{i-1} \leq \alpha < q_i$, for some i. In this case the edge is chosen to be i.)

Recall that the judge was required to always use the same edge for the same contract. (The reader may note that failing to do so may creates conflicting verdicts; and furthermore, give advantage to parties who keep appealing to court many times with the same contract.) Using the same edge for the same contract can be accomplished if the judge operates as follows. Once he chooses an edge e with respect to a contract $CONT$, he stores the pair $(e, CONT)$, and uses e again any time that the $CONT$ is brought before him. This solution can hardly be considered efficient, as the judge must keeps record of all previous verdicts. It is our goal to free the judge from any bookkeeping. We do this by using the "poly-random functions" of Goldreich Goldwasser and Micali [GGM].

In [GGM] it is shown how to use any one-way (in a very weak sense [L]) function to efficiently construct a set of functions that are "very few" in number, deterministic and fast computable, but *possess all the statistical properties of truly random functions with respect to an observer with polynomially bounded resources.* Their deterministic construction, given as input a constant c and a randomly chosen n-bit long string r, outputs a determinstic, polynomial-time algorithm f_r. This algorithm, on input a n-bit long string x, outputs a n^c-bit long string $f_r(x)$. These functions (deterministic algorithms) f_r's cannot be distinguished from random functions by any probabilistic polynomial time algorithm that asks and receives the value of a function at arguments of its choice. (Here by a random function, we mean a function randomly selected with uniform probability from the set of all functions from n-bit strings to n^c-bit strings.)

Let us now show how to use this result in order to free the judge from bookkeeping. The judge randomly selects a n-bit string r once and for all. No other random choices are ever done by the judge. When invoked with $CONT$ and all other inputs, the judge computes $f_r(CONT)$ and uses it (instead of flipping coins) to determine ths choice of the edge. This way of operating essentially maintains μ-fairness. It can be shown that the conditional probability that "A has B's signature on $CONT$" given that "B has A's signature on $CONT$" is greater than $1 - 1/k - 1/n^c$, for all constants c and sufficiently large n. (This follows from the properties of the poly-random functions, else all functions easy to evaluate can also be easily inverted.) We stress that the μ-faireness of protocol holds even if the parties are given the edges relative to other contracts of their choice.

In the above we assume n to be large enough so that all contracts have length smaller than n. There is no need to assume so: it is sufficient that each contract starts with an n-bit string C uniquely associated to it. Half of the bits of C are chosen by A and the remaining half by B. As long as one party never uses the same bits for another contract, then no matter how "trickly" the second party will choose the second half of C, the same security is maintained. This greatly improves the efficiency of the scheme.

Why poly-random functions?

Poly-random functions improve previous results of Blum and Micali [BM] and Yao [Y] on pseudo-random number generators. They present deterministic algorithms that transform a truly random (but short) secret seed to a long pseudo-random bit-sequence passing polynomial time statistical tests. Such high quality pseudo-random number generators may not be straightforwardly applied in our context. For example

1) Using *CONT* or part of it as input to the generator does not work. The parties may also do so and determine the edge as well.

2) Using the bit by bit "exclusive or" of *CONT* and some fixed key r (randomly and secretly selected by the judge) as input to the generator, may not work. In fact, these generators are proved to produce "random" outputs only on randomly and independently selected inputs. In our setting, knowledge of the edges relative to previous contracts, may help in predicting the edge relative to a new contract.

Remark: Above, we have suggested that the edge be determined by the judge. An alternative method for determining the edge (i.e. determining a fraction in the interval [0,1]) is described below. Each party has a one-way function associated with him. At step 1 of the protocol, each party chooses randomly a fraction in the interval [0,1], applies his one-way function to the binary extension of this fraction and appends the result to his declaration (which is only then signed). The fraction (which determines the edge) is agreed to be the sum reduced modulo 1 of these fractions. This way, the edge will be determined during the first step of the protocol, by both parties. However, none of them knows at that point what the edge is.
Although this solution appeals to be more elegant, it suffers from a serious drawback. In order to rule, the judge must find out the fractions chosen by both parties. This requires either that both parties appear in court or that the judge knows to invert the one-way functions of all users. Both requirements are highly undesirable from a practical point of view. Note that when the edge is chosen by the judge, the judge is able to rule even if only one party appears in court.

2.4 Summary and Further Discussion

Let us first sum up the properties of our solution (which consists of a two-party protocol and a judge-procedure):

1) It satisfies both the viability and the fairness requirements, *without using* the "Equal Computing Power" assumption. It is not prone to "early stopping". Furthermore, unlike other solutions, our solution requires the intervention of a trusted third party and a "time out" mechanism in a very mild sense.

2) A third party (a judge) intervenes only in case of dispute. In this case the judge can rule even if only one party appears in court. Furthermore, no bookkeeping is required from the judge!

3) The protocol is *optimal* in the sense that it can be implemented and proven fair under the minimum possible intractability assumption: the existence of one-way functions and secure signature schemes. (The judge-procedure which involves the use of random functions requires a slightly stronger assumption [GGM].)

4) The protocol is *optimal* in the sense that it uses the minimum number of message exchanges needed to satisfy the μ-fairness requirement.

5) The protocol is very easy to execute. Also the judging-procedure is conceptually simple and only requires the examination of *two* messages (the signed initial declaration and the last received secret).

Acknowledgments

We wish to thank Shimon Even and Adi Shamir for very illuminating discussions.

References

[B] Blum, M., "How to Exchange (Secret) Keys", *ACM Trans. on Comp. Sys.*, Vol. 1, No. 2, 1983, pp. 175-193. Also in the *Proc. of the 15th ACM Symp. on Theory of Computation*, 1983, pp. 440-447.

[BM] M. Blum and S. Micali, "How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits", *SIAM Jour. on Computing*, Vol. 13, Nov. 1984, pp 850-864 (Preliminary version: Proc. 23rd IEEE Symp. on Foundations of Computer Science, 1982, pp 112-117.)

[BR] Blum, M., and Rabin, M.O., "Mail Certification by Randomization", in preparation.

[DH] Diffie, W., and Hellman, M.E., "New Directions in Cryptography", *IEEE Trans. on Inform. Theory*, Vol. IT-22, No. 6, November 1976, pp. 644-654.

[E] Even, S., "A Protocol for Signing Contracts", TR No. 231, Computer Science Dept., Technion, Haifa, Israel, 1982. Presented in *Crypto81*.

[EGL] Even, S., Goldreich, O., and Lempel, A., "A Randomized Protocol for Signing Contracts", *Advances in Cryptology: Proceedings of Crypto82*, (Chaum D. et. al. eds.), Plenum Press, 1983, pp. 205-210. A better version will apear in the *Comm. of the ACM*.

[EY] Even, S., and Yacobi, Y., "Relations Among Public Key Signature Systems", TR No. 175, Computer Science Dept., Technion, Haifa, Israel, 1980.

[G] Goldreich, O., "A Simple Protocol for Signing Contracts", in *Advances in Cryptology: Proceedings of Crypto83*, (Chaum D., ed.), Plenum Press, pp. 133-136, 1984.

[GGM] Goldreich, O., Goldwasser, S., and Micali, S., "How to Construct Random Functions", *Proc. of the 25th IEEE Symp. on Foundation of Computer Science*, 1984, pp. 464-479. To appear, *Journal of ACM*

[GMR] Goldwasser, S., Micali, S., and Rivest, R.L., "A *Paradoxical* Solution to the Signature Problem", *Proc. of the 25th IEEE Symp. on Foundation of Computer Science*, 1984, pp. 441-448.

[HS] Hastad, J., and Shamir, A., "The Cryptographic Security of Truncated Linearly Related Variables", to appear in the proceedings of the *17th STOC*, 1985.

[L] Levin, L.A., "One-way Functions and Pseudorandom Generators", to appear in the proceedings of the *17th STOC*, 1985.

[R] Rabin, M.O., "Transaction Protection by Beacons", TR-29-81, Aiken Computation Laboratory, Harvard University, 1981.

[RSA] Rivest, R.L., Shamir, A., and Adleman, L., "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", *Comm. of the ACM*, Feb. 1978, pp. 120-126.

[Y] Yao, A.C., "Theory and Application of Trapdoor Functions", *Proc. of the 23rd IEEE Symp. on Foundation of Computer Science*, 1982, pp. 80-91.