J. Cryptology (1995) 8: 201–222

Journal of
CRYPTOLOGY
© 1995 International Association for Cryptologic Research

Practical and Provably Secure Release of a Secret and Exchange of Signatures

Ivan Bjerre Damgård
Mathematical Institute, Aarhus University,
Ny Munkegade 116, DK-8000 Aarhus C, Denmark

Communicated by Gilles Brassard

Received 20 October 1992 and revised 11 July 1994

Abstract. We present a protocol that allows a sender to release gradually and verifiably a secret to a receiver. We argue that the protocol can be efficiently applied to the exchange of secrets in many cases, such as when the secret is a digital signature. This includes Rabin, low-public-exponent RSA, and El Gamal signatures. In these cases, the protocol requires an interactive three-pass initial phase, after which each bit (or block of bits) of the signature can be released noninteractively (i.e., by sending one message). The necessary computations can be done in a couple of minutes on an up-to-date PC. The protocol is statistical zero-knowledge, and therefore releases a negligible amount of side information in the Shannon sense to the receiver. The sender is unable to cheat, if he cannot factor a large composite number before the protocol is completed.

Key words. Exchange of secrets, Digital signatures, Zero-knowledge.

1. Introduction

1.1. *The Basic Problem*

Suppose parties A and B each possess a secret, s_A and s_B, respectively. Suppose further that both secrets represent some value to the other party, and that they are therefore willing to "trade" the secrets against each other. For example, s_A might be A's digital signature on a commitment to deliver some kind of service to B, while s_B could be a bank's signature on some digital cash. However, if the parties do not trust each other, it is clear that none of them is willing to go first in releasing the secret—once one of them has done this, he may never get anything in return.

If the two secrets are represented as bit strings of the same length, this can be solved by exchanging the secrets bit by bit; if this is done honestly, no party will be more than one bit ahead of the other. Put another way: if at some point, A can compute s_B in time T, then B can compute s_A in at most time $2T$ by

guessing the bit he may be missing. This assumes, of course, that a bit of s_B tells A just as much as a bit of s_A tells B—we get back to this problem in Section 2.2.

However, this "solution" has created another problem: one party may have given away his secret, only to find in the final stage that in return he has been given garbage instead of bits of a genuine secret. Hence what we need is a way to release the secrets in small parts, such that the receiver can *verify* for each part that he has been given correct information. The alternative, namely to assume a trusted third party which would need to be on-line and active in every instance of the protocol, is not attractive and probably not very realistic either.

Thus we can distill a basic primitive (introduced in [7]) which we call *gradual and verifiable release of a secret*, the intuitive meaning of which should be clear from the above. It should also be clear that a gradual and verifiable release protocol can be used to implement an exchange of secrets between any number of parties. In order for such an exchange to be *fair*, the secrets involved have to satisfy certain conditions (see Section 2.2). In addition, the concept of a release protocol makes sense in its own right, and might be useful for other purposes than implementing exchanges of secrets.

1.2. *Comparison with Earlier Work*

The exchange and release of secrets has attracted a lot of attention in the past, and a large body of literature exists on the subject [3], [5], [7], [8], [11], [12], [18]–[22], [24], [27]. A general solution to the release of secrets problem follows from the discovery of zero-knowledge proofs and arguments for any NP-language [6], [14], because these techniques can be used to construct a release protocol that is as secure as the bit-commitment scheme used in the zero-knowledge proof. Thus existence of any one-way function is a sufficient assumption to implement a secure gradual release. This is relatively trivial to see for methods that release specific bits of the secret, while more advanced methods are required to release probabilistic information, which can amount to less than one bit at a time (see, for example, [20] and [15]). It is even possible to make the choice of bits to release adaptive [19]. In this work we construct a simple bit-by-bit release, which can be used directly, or serve as a primitive in constructions aiming at releasing "parts of bits."

The gradual release protocols resulting from the general theoretical results are very far from practical, however, and a solution that is both practical and provably secure does not seem to have appeared before.

Before looking at the basic problem with earlier practical solutions, we have to point out a fundamental fact: the demand that the released parts of the secret be correct makes sense only if the secret is itself determined by some public piece of information. Otherwise, not even the secret itself can be verified. Typically, something like $f(s)$ is public, where s is the secret and f is some one-way function.

Earlier practical release protocols assume *a priori* that the secret is given in some particular form, e.g., a discrete log in [7], a factorization in [5]. However, when trying to apply such protocols, we are likely to find that the application

dictates the way in which the secret is given, i.e., the particular one-way function f involved is determined by the application. For example, if we want to release an RSA signature on a given message, f would be the function mapping a signature to the message it signs. Thus, if we wanted to use, e.g., [7], we would have to make both $f(s)$ and g^s known, where g is chosen in some appropriate group. The problem is that this may release additional information about s, and so is very unlikely to lead to a provably secure scheme.

The release protocol of this paper solves the problem by using a new tool: an unconditionally hiding bit commitment scheme that allows commitment to a string of any length and can be opened bit by bit. Moreover, commitments have length independent of the strings they hide. In addition, we present efficient protocols for checking that the contents of such a commitment has a particular form (for example, that it is the Rabin, RSA, or El Gamal signature on a given message). This leads to provably secure release protocols for such signatures.

Since digital signatures are obvious candidates for representing value or commitments in practical applications, methods for releasing or exchanging such signatures seem to be of very strong practical relevance.

1.3. *Fair Exchange and Contract Signing*

Intuitively, a fair exchange of secrets protocol is one that avoids a situation where A can obtain B's secret, while B cannot obtain that of A. If there is no assumption made that third-party intervention is possible, the best we can do is to guarantee that if one party stops the protocol early, both parties are left with roughly the same computational task in order to find the other party's secret. This is the model used in this paper. In Section 2.2 we discuss to what extent such a fair exchange follows from a gradual and verifiable release.

However, in any case, it is clear that if, for example, A has much more powerful hardware than B, the *actual time* A would need to find the secret in case of early stopping would be much smaller than for B. Depending on the application this may or may not be a problem. One example where this comes up is if we use exchange of secrets to implement fair contract signing. This is straightforward: A and B both sign the contract, and then gradually exchange their signatures. However, if the contract involves time-related issues, such as a commitment taking effect at a certain date, the above "real-time" problem could be serious in case one party cheats.

In [2] Ben-Or *et al.* show how to avoid this problem, if we assume that a judge is available to settle disputes. Since the judge only has to be active in the event of a dispute, this assumption is less demanding than one that calls for an active, on-line trusted third party. The protocol can make use of any signature scheme. The difference to our work is first the assumption about the judge, and second that the protocol of [2] is a dedicated protocol for solving the contract-signing problem: it does not implement an exchange or a release of secrets.

The protocol in [2] involves a certain computational overhead: the signature scheme must be employed a large number of times by both parties. In [10] we showed that most of this overhead can be avoided.

1.4. *Organization of the Paper*

In Section 2 we give a formal definition of release protocols, and a discussion of the extent to which release protocols can be used to build exchange protocols both in general and in the particular cases we consider. The reader most interested in the actual protocol constructions can skip this section in a first reading. However, note that some of the material in Section 2.2 can be read before Section 2.1 and may be of interest independently of the formal definitions.

Section 3 describes the bit commitment scheme we base our protocols on and proves its basic properties, while Section 4 gives some protocols for checking the contents of commitments. Finally Sections 5 and 6 describe complete protocols for releasing RSA/Rabin signatures and discrete log-based signatures, respectively.

2. Basic Definitions

This section gives some basic definitions and some connections between them. Although the model is certainly not the most general possible, it does describe appropriately the protocols we present in the following.

2.1. *Release Protocols*

In this section we give a formal definition of a secure release protocol. Intuitively, we model the situation where party A has a secret s, which he will release bit by bit to B who knows t, where (if A is honest) (t, s) satisfy some predicate P. Typically, P will be satisfied if t is the image under some one-way function of s. At any point in the protocol, B should be able to compute some of the bits of s correctly, but the protocol should not give him more than this, i.e., he should be in the same situation as if he had been given t and the bits of s by an oracle. Note that, in general, B may be able to compute (on his own) additional bits of s from ones already released. This is of no concern in this definition, however, because all we want to express is that, at any point of the protocol, B gets whatever he can compute efficiently from the information he is entitled to know at the given time.

We think of the pair (A, B) as interactive Turing machines as defined in [16] and use the notation of [13]. In particular, both A and B are polynomial-time bounded in the input length, and are equipped with knowledge tapes containing their private inputs. In the following X means A or B. Following [13], we let $\bar{X}$ denote a machine following the protocol specified for party X, while $\tilde{X}$ denotes an arbitrary cheating participant playing the role of X. The symbol X represents $\bar{X}$ or $\tilde{X}$.

A bit string t of length k is common input to A and B, while A receives as private input on its knowledge tape a string s of length at most $f(k)$, where f is a polynomial. To model the fact that B may have some *a priori* knowledge, we assume that B receives the string k_B on its knowledge tape.

In the following, $s|_i$ denotes the i least-significant bits of s ($s|_0$ is the empty string).

The event that one party sends a message to the other is called a *pass*. Passes are numbered ordinarily, starting from 1. After each pass, the participant receiving a message may output "reject" and stop, indicating detection of cheating. We say that (A, B) *completes pass* i if no party outputs reject after pass i.

Following [13], the view of a participant is defined to be the ordered concatenation of the messages sent in the protocol, followed by the random bits read by the participant. This is denoted by $View_X(t, s, r_A, k_B, r_B)$, where r_X is the content of the random tape of party X. The symbol $View_X^i(t, s, r_A, k_B, r_B)$ denotes party X's view of the truncated protocol where we only consider passes 1 through i (note that this view may be shorter than i passes if the protocol stops earlier). In the following, $View_{\bar{B}}(\cdots)$ always refers to a conversation with A, while $View_B(\cdots)$ refers to a conversation with $\bar{A}$.

To define the properties of (A, B), certain items need to be specified. We have to be able to tell which values of A's secret are correct, and which are not; it should be specified at which points in time the bits of s are released; and finally a method has to be specified which B can use to compute the bits of s from the messages he sees. Formalizing this, we define the properties of (A, B) with respect to the following:

- A fixed polynomial-time computable predicate P. It takes as input two of the strings defined above: the k-bit string t and the bit string s of length at most $f(k)$. The meaning of P is that it tests whether s is a "correct" secret with respect to the common knowledge t.
- A series of increasing functions $\{p_k\}_{k=1}^{\infty}$, where

$$p_k: \{0 \cdots f(k)\} \to N,$$

  and where $p_k(f(k))$ is polynomially bounded. The meaning of p_k is that, for input length k, $p_k(i)$ is the index of the first pass after which the protocol enables B to compute $s|_i$.
- A set of polynomial-time computable functions

$$\{h_k^i \mid k = 1..\infty, i = 1..f(k)\}$$

  such that h_k^i takes as input a sample of $View_B^{p_k(i)}(t, s, r_A, k_b, r_B)$. As output, it produces an i-bit string. These functions should be used by B to compute the first i bits of the secret after pass $p_k(i)$ is completed. The value $h_k^i(View_B^{p_k(i)}(t, s, r_A, k_b, r_B))$ is said to be *correct* if there is a z of length at most $f(k)$ bits such that $P(t, z) = 1$ and

$$z|_i = h_k^i(View_B^{p_k(i)}(t, s, r_A, k_B, r_B)).$$

  Otherwise it is incorrect.

We can now give the following:

Definition 1. The pair (A, B) is called a *secure release protocol with respect to* P, $\{p_k\}$, *and* $\{h_k^i\}$ if the following three properties are satisfied:

1. If A and B follow the protocol, B always gets correct information. Formally: If A and B follow the protocol, then, for any (t, s) such that $P(t, s) = 1$, any $i = 0..f(|t|) = f(k)$, and any r_A, k_B, r_B, we require that $h_k^i(View_B^{p_k(i)}(t, s, r_A, k_B, r_B))$ is correct.
2. A cheating A should be able to convince B about incorrect bits of s with only negligible probability. We define this by saying that the event that B computes an incorrect value for the first i bits of s almost never happens simultaneously with the event that no cheating is detected in the first $p_k(i)$ passes. Formally: $\forall A\ \forall c\ \exists k_0\ \forall |t| > k_0\ \forall s, r_A, k_B\ \forall i = 1..f(k)$:

$$Prob\Big(h_k^i\big(View_B^{p_k(i)}(t, s, r_A, k_B, r_B)\big) \text{ incorrect}$$

$$\textbf{and } (A, \bar{B}) \text{ completes pass } p_k(i)\Big) \leq k^{-c}.$$

   The probability is taken over the choice of r_B.
3. At each point of the protocol, B should know no more than what he can compute from the information he is entitled to know at that point. This is defined by requiring that B can simulate the conversation up the given point from only the information he is supposed to know. Formally:

   For each B, an expected polynomial-time machine M_B exists, which on input bit strings x (first i bits of a correct secret), t, k_B and with random tape r_M simulated B's view of the first $p_k(i)$ passes of the conversation with $\bar{A}$. Let $M_B(x, t, k_B, r_M)$ denote M_B's output, considered as a random variable with distribution taken over r_M.

   We then require that, for all k and all $i = 0 \cdots f(k)$, whenever $x = s|_i$ for some s with $P(t, s) = 1$, the distribution of $M_B(x, t, k_b, r_M)$ is statistically indistinguishable from that of $View_B^{p_k(i)}(t, s, r_A, k_B, r_B)$, where the distribution is taken over r_A and r_B.

Remarks

- For simplicity, we only consider a bit-by-bit release in the above definition. The definition could trivially be generalized to talk about a release of a block of bits per pass.
- We need to be able to simulate for each $i = 0..k$, because we want A to be protected, even if $\bar{B}$ stops before all bits are released. The best we can do in such a case is to require that $\bar{B}$ can compute only what he can get from the information he is entitled to know at the given time.

2.2. *Exchange Protocols*

In this section we discuss to what extent release protocols can be used to build fair exchange protocols. Suppose parties X and Y possess secrets s_X, s_Y,

respectively, defined by (possibly) different predicates P, P'. Then if we have release protocols for these predicates as in Definition 1, it is natural to try to exchange the secrets by interleaving the release of s_X with that of s_Y.

Consider now the question whether this exchange protocol is fair, where we think of fairness as defined by Yao [27]: even a cheating Y (or, symmetrically, a cheating X) cannot force a situation where it is feasible for him to find s_X, but infeasible for X to find s_Y.

It is clear from Definition 1 that the interleaving approach forces the parties to send correct bits of their secrets, and also that each party knows at each point only a prescribed number of bits of the opponent's secret. Nevertheless, the exchange will not necessarily be a fair one *in general*: it is possible that, for example, s_X is easily computed already from the first half of its bits. If this is not the case for s_Y, then Y could gain an unfair advantage by quitting halfway through the protocol, perhaps leaving X with only useless information about s_Y.

The point is of course that the problem Y has to solve to find s_X may be of a totally different nature than the one X is facing to find s_Y. The parties may of course always decide to use the interleaved exchange anyway if they believe that the two particular problem instances involved have similar complexities. However, to say something more general, we have to be restricted to a set of "nicer" cases, where it is possible to connect the two problems. One possible step in this direction is to require that s_X, s_Y be defined by the same predicate (i.e., $P = P'$), and that the corresponding public strings t_X, t_Y be drawn independently from the same distribution. This leads to the following definition of the exchange protocol *induced* by a release protocol:

Definition 2. Let (A, B) be a release protocol secure with respect to P, $\{p_k\}$, and $\{h_k^i\}$ (see Definition 1). The following two-party protocol (X, Y) is called *the exchange protocol induced by* (A, B):

X and Y receive two common inputs t_X, t_Y, both of length k bits and drawn independently from the same probability distribution π_k. X and Y get as private input s_X (resp. s_Y), such that $P(t_X, s_X) = P(t_Y, s_X) = 1$.

X simulates copies A_X, B_X of A and B, giving s_X, t_X as input to A_X and t_Y as input to B_X. Correspondingly, Y runs copies A_Y, B_Y on inputs s_Y, t_Y and t_X. X, Y will now, for $i = 1, 2, \ldots$, execute pass i of (A_X, B_Y) followed by pass i of (A_Y, B_X), until both protocols halt.

Even an induced exchange protocol is not guaranteed to be fair if we do not know anything about the predicate P: it is possible that, for a nonnegligible fraction of the t's, finding an s, such that $P(t, s) = 1$, is much easier than for other t values. If t_X happens to be such an easy case, Y is clearly in a better situation than X. What we need to avoid this is that the problem of finding s such that $P(t, s) = 1$ based on t and some bits of s is of about the same difficulty for nearly all choices of t under π_k. One way of stating such "uniform hardness" of a problem a little more precisely is to say that any algorithm that solves a nonnegligible fraction of the instances of the problem can be turned into an algorithm that uses not much more time, and solves nearly all instances.

With this assumption on P, we can say the following about the induced exchange: assume that some $\tilde{Y}$ has a strategy for aborting the protocol at some stage and subsequently finding s_X with some nonnegligible probability. When the protocol is aborted, this leaves $\tilde{Y}$ with t_X and, say, i bits of s_X. X is left with t_Y and i or $i-1$ bits of s_Y. This is essentially the *only* information the parties have been given since, from it, the entire views of X (resp. $\tilde{Y}$) can be simulated, by condition 3 of Definition 1. So except for perhaps one bit X has to guess, this means that both parties are faced with samples of the same problem, drawn from the same distribution. By the above assumption on P, this implies that whatever method $\tilde{Y}$ uses to find s_X will also work for X to find s_Y, and therefore the protocol is fair.

Using the simulators guaranteed by Definition 1, this reasoning can be formalized. One way of doing this is shown in the Appendix. Here we concentrate on the question of whether the types of secrets one might want to exchange in practice are likely to have a uniform hardness property such as the one we have discussed.

We have already discussed that digital signatures are interesting in this context. So, as an example, assume that t specifies a message and an RSA public key, and that $P(t, s) = 1$ precisely if s is a valid RSA signature on the message. With our current knowledge, we can only conjecture that this predicate has an appropriate uniform hardness property. Some evidence is known in favor of this conjecture, however: from the multiplicative property of RSA, it follows easily that if you can sign in polynomial time a polynomial fraction of the messages for some modulus, then you can sign all messages using that modulus in expected polynomial time. Moreover, the results of [1] give strong indications that something similar holds when some number of bits of s are given. This alone is not enough to argue uniform hardness for RSA. Since presumably X and Y will be using different moduli, we also need to know that signatures generated from different moduli are equally hard to forge. Thus the key-generation algorithm must ensure that no significant fraction of the moduli generated are much easier to break than others—this of course is a natural user requirement, even if the signatures are not being used in exchange protocols.

Since El Gamal signatures are known to have properties similar to the multiplicative properties of RSA, we find it reasonable to conjecture that at least the signature schemes we consider in this paper have uniform hardness sufficient to make induced exchange protocols fair when using these signatures.

At this point one could perhaps complain that the assumptions made in the definition of induced exchange protocols, and particularly the assumption that t_X and t_Y are identically distributed, are too demanding in practice. What if Y could somehow manipulate the distribution of t_X and/or t_Y, presumably to make life easier for himself? Whether this objection is reasonable or not will of course depend on the practical circumstances. However, there is one general remark we can make: if messages are hashed before they are signed—as is nearly always the case in practice—Y is not likely to benefit from manipulating messages: if the hash function used is strong, he will not be able to control the hash result and, for example, force t_X to be an easily signed hash value (of

which there are only very few). This is the same kind of reasoning that underlies the Fiat–Shamir signature scheme.

In summary, we have argued that exchange protocols induced from release protocols are useful in many cases that are important in practice. As a side remark, it is also worth noting that more complicated exchange protocols that can deal with seemingly incompatible types of secrets typically work by choosing some auxiliary secret w, making public some information connecting w and the actual secrets, and then releasing w bit by bit, see, e.g., [27]. Thus a bit-by-bit release as defined here can also be useful as a building block in other protocols.

3. The Bit Commitment Scheme

In this section we define the bit commitment scheme we use, and prove its basic properties.

To set up the commitment scheme, B, who will receive commitments from A in the following, must generate and make public a Blum integer [26], [4] N (i.e., $N = pq$, where p, q are primes congruent to 3 modulo 4), and g, a quadratic residue modulo N chosen uniformly at random. The security for B will be based on the difficulty of computing a square root modulo N of g. Therefore the uniform random choice of g is essential: it is well known that this ensures that computing a square root of g is equivalent to factoring N. In the following the commitment scheme is used as a tool in release protocols satisfying Definition 1. The bit length of N should therefore be polynomially related to the security parameter k of these protocols.

Having chosen N and g, B must in zero-knowledge prove that he knows the two prime factors of N [23], prove that they are both congruent to 3 modulo 4[1] [17], and that g is a quadratic residue [16]. The methods for doing this are well known and quite efficient, and in any case this step does not have to be executed at every application of the commitment scheme.

In this first phase either A acts as the verifier or this role is played by a trusted third party. The latter case is the most likely one in practice, as setting up a large-scale public-key system nearly always requires a certification authority that registers users and certifies the relation between identities and public keys (moduli). Such a center might as well act as the verifier in the above, and certify by a digital signature that N and g have been verified successfully.

Let $SQ(N)$ denote the subgroup of quadratic residues modulo N. Having established N and g, the parties agree on a natural number l. Then A can commit to any integer s satisfying $-2^{l-1} < s < 2^{l-1}$ by choosing R uniformly at random in $SQ(N)$ and computing the commitment

$$BC_g(R, s) := R^{2^l} g^s.$$

[1]Actually, the protocol in [17] only proves that $N = p^r q^s$, where r, s are odd and p, q are 3 modulo 4. However, even for such numbers, squaring is a permutation of the quadratic residues, and this is the property we need.

This is called a base-g commitment. A commitment is opened by revealing R and s, which allows B to verify the above equation.

The commitment scheme is based on the hash functions from [9]. In fact, $BC_g(R, s)$ is precisely the hash value of s computed with starting point R, using the factoring-based hash function from [9]. The same type of function was used in [25] for the purpose of fail-stop signatures.

In this scheme, A is prevented from cheating by the difficulty of finding collisions for the function BC_g, while the security against B is ensured by the uniform choice of R done by A. More precisely, we have the following lemma:

Lemma 1. *The commitment $BC_g(R, s)$ has distribution independent of s, when R is a uniformly chosen square* mod N. *Moreover, if A can open the same commitment using values R, s (resp. R', s'), where $s \neq s'$, then A can compute a square root modulo N of g.*

Proof. The first statement is clear from the fact that squaring modulo a Blum integer is a permutation, and that therefore a commitment is always a uniformly chosen element in $SQ(N)$. For the second statement, assume without loss of generality that $s' > s$ and write $s' - s = (2h + 1)2^j$. Clearly, $j < l$. Then the equation

$$R^{2^l} g^s = R'^{2^l} g^{s'} \bmod N$$

implies that

$$(R/R')^{2^{l-j}} = g^{2h+1} \bmod N$$

and therefore $(R/R')^{2^{l-j-1}} g^{-h}$ is a square root of g. □

These properties of the function BC_g were also used in [9]. The crucial property in this context, however, is that these commitments can be opened *gradually*: given a commitment $BC_g(R, s)$ to a positive number s, A can reveal the least-significant bit b of s by revealing X such that

$$X^2 \bmod N = BC_g(R, s) \qquad \text{if} \quad b = 0$$

and

$$g \cdot X^2 \bmod N = BC_g(R, s) \qquad \text{if} \quad b = 1.$$

After this, X can be regarded as a commitment to $s/2$ (with l replaced by $l - 1$) and more bits of s can be opened.

By essentially the same argument as in Lemma 1, it is easy to see that if A knows how to open in one step the entire value of $s > 0$, he cannot open single bits of s with values that are inconsistent, unless he can compute a square root of g.

It is also clear that the procedure for opening one bit can easily be generalized to allow opening in one step of any number of the least-significant bits of s.

Since computing square roots of random numbers mod N is equivalent to factoring N, we need the following assumption about the hardness of factoring:

Factoring Assumption. A probabilistic polynomial-time algorithm Δ exists which on input 1^k outputs a k-bit Blum integer N, such that, for any probabilistic polynomial-size circuit family C, and any constant c, the probability that C factors N is at most k^{-c}, for all sufficiently large k. This probability is taken over the random choices of Δ and C.

4. Checking the Contents of Commitments

When A sends a commitment as above, there is no reason *a priori* to believe that this represents anything useful: A may not even know how to open the commitment he sends. We therefore need the following protocol, which is based on the proof system from [7], and allows us to check that A knows how to open a commitment, and furthermore that the opening will reveal a number in a given interval.

We let the interval be $I =]a \cdots b]$ and let $e = b - a$. We define $I_{\pm e} =]a - e \cdots b + e]$. These parameters must be chosen such that $I_{\pm e}$ is contained in the legal range for openings of commitments $]-2^{l-1} \cdots 2^{l-1}[$. The protocol will be secure for A—in fact statistical zero-knowledge—if he knows how to open a given commitment $c = BC_g(R, s)$ to reveal $s \in I$. Moreover, it will convince B that $s \in I_{\pm e}$.

PROTOCOL CHECK COMMITMENT
This protocol is executed with commitment $c = BC_g(R, s)$ as input to A and B. It is secure for A if he knows how to open c to reveal $s \in I$, and will convince B that $s \in I_{\pm e}$.

Execute the following k times in parallel:

1. A chooses t_1 uniformly in $]0..e]$, and sets $t_2 = t_1 - e$. He sends the unordered pair of commitments $T_1 = BC_g(S_1, t_1)$, $T_2 = BC_g(S_2, t_2)$ to B.
2. B requests to see one of the following:
   (a) Opening of both T_1 and T_2.
   (b) Opening of $c \cdot T_i$ mod N, where A chooses i such that $s + t_i \in I$.
3. In the first case of step 2, B checks that both numbers opened are in $]-e..e]$, and that their difference is e. In the second case, B checks that the number opened is in I.

B outputs reject and stops if any of the openings are not correctly done, or if any of the checks required are not satisfied.

The properties of this protocol are summarized in the following two lemmas. Intuitively the first one says that A can only cheat with negligible probability, and the second says that if A behaves correctly, B learns nothing except the fact that $s \in I_{\pm e}$.

Lemma 2. *Given correct answers to both* (a) *and* (b) *in one instance of steps* 1–3 *above, a pair* R, s *such that* $s \in I_{\pm e}$ *and* $c = BC_g(R, s)$ *can be computed efficiently.*

Proof. By assumption, we are given X, x, Y, y such that

$$T_i = X^{2^l} g^x \quad \text{and} \quad c \cdot T_i \bmod N = Y^{2^l} g^y,$$

where $x \in]-e..e]$ and $y \in I$. These two equations imply that we can write c in the form $c = BC_g(Y/X \bmod N, y - x)$ so that the result follows from putting $R = Y/X \bmod N$ and $s = y - x$. □

Lemma 3. *Knowing the factorization of* N, *any* B*'s view of CHECK COMMITMENT when talking to* $\bar{A}$ *can be simulated perfectly, provided* $\bar{A}$ *is given an* s *in* I.

Proof. With the factorization of N, modular square roots are easy to compute, and so given a square Q, for any s, we can compute R, such that $Q = BC_g(R, s)$. Armed with this observation, the simulation is quite trivial: we simply generate all the unordered pairs T_1, T_2 as random squares and send them to B. If, for a given pair, we get request (a) from B, we choose t_1, t_2 as A would have done, and open T_1, T_2 accordingly. If we get request (b), we choose i at random to be 1 or 2, choose a random $x \in I$, and open $c \cdot T_i$ to reveal x. The simulation of case (b) works since, in the real conversation, $s + t_i$ is always a uniformly chosen number in I, independent of s (provided $s \in I$). □

A slight variant allows us to show that two commitments c, c' contain the same number, even if the commitments use different bases, say g and h:

PROTOCOL COMPARE COMMITMENTS

This protocol is executed with commitments c, c' as input to A and B. It is secure for A if he knows how to open both c and c' to reveal $s \in I$, and will convince B that c and c' contain the same value $s \in I_{\pm e}$.

Execute the following k times in parallel:

1. A chooses t_1 uniformly in $]0..e]$, and sets $t_2 = t_1 - e$. He sends to B the unordered pair $((T_1, T_1'), (T_2, T_2'))$, where each component of the pair is ordered and is defined by $(T_i, T_i') = (BC_g(S_i, t_i), BC_h(S_i', t_i))$.
2. B requests to see one of the following:
   (a) Opening of (T_i, T_i') for both $i = 1$ and 2.
   (b) Opening of $c \cdot T_i \bmod N$ and $c' \cdot T_i'$, where A chooses i such that $s + t_i \in I$.
3. In the first case of step 2, B checks that opening T_i and T_i' has resulted in the same number, that both numbers opened are in $]-e..e]$, and that their difference is e. In the second case, B checks that opening $c \cdot T_i \bmod N$ and $c' \cdot T_i'$ reveals the same number, and that this number is in I.

B outputs reject and stops if any of the openings are not correctly done, or if any of the checks required are not satisfied.

The following two lemmas give the basic properties of this protocol. Their intuitive meaning and proofs correspond exactly to those of Lemmas 2 and 3.

Lemma 4. *Given correct answers to both* (a) *and* (b) *in one instance of steps* 1–3 *above,* R, R', *and* s *such that* $s \in I_{\pm e}$, $c = BC_g(R, s)$, *and* $c' = BC_h(R', s)$ *can be computed efficiently.*

Lemma 5. *Given the factorization of* N, *any* B*'s view of COMPARE COMMITMENTS when talking to* $\bar{A}$ *can be simulated perfectly, provided* $\bar{A}$ *is given an* s *in* I.

5. Release of Rabin and RSA Signatures

We are now ready to present a complete protocol for the release of a Rabin signature, which is essentially an RSA signature where the public exponent is fixed to 2. Hence the public key in this signature scheme is simply a modulus n. This number should not be confused with the modulus N, which is used in the bit commitment scheme.

The common input to the parties is n and a message $m \in]0 \cdots n[$, while A's private input is the signature on m using public key n. Usually, such signatures are represented by numbers between 0 and n, but for technical reasons we prefer the interval from n to $2n$, i.e., we assume that the honest A knows a number s in $]n..2n]$ such that $s^2 \bmod n = m$. Thus k, the length of the common input, is $2|n|$, where $|n|$ is the bit length of n. For all commitments in the following, we use $l = 2|n| + 3$.

We now show the protocol, and then state its properties formally.

PROTOCOL RELEASE RABIN SIGNATURE

This protocol is executed with common input n, m to A and B and private input s to A. Informally speaking, the protocol convinces B that he receives bits of a value $s \in]0..3n]$, such that $s^2 \bmod n = m$, and A is ensured that at each stage B knows only a prescribed number of bits of s.

1. B chooses the parameters of the bit commitment scheme N, g. These are verified interactively as explained in Section 3.
2. A sends to B the commitment $h = BC_g(R, s)$.
   A sends $v = BC_h(R', s)$ and $w = BC_g(R'', d)$, where d is defined by $s^2 = m + dn$.
   A opens (as a base-g commitment) the product $g^m w^n v^{-1} \bmod N$ to reveal a 0. Note that if h, v are constructed correctly, then $v = BC_g(R'R^s, s^2)$.
3. A uses the CHECK COMMITMENT protocol with $I =]n - 1 \cdots 4n - 1]$ to prove that he knows how to open w to reveal a value d in
   $$]-2n - 1..7n - 1].$$
   A uses the COMPARE COMMITMENTS protocol with $I =]n \cdots 2n]$ to prove that he knows how to open h and v to reveal the same value s, and that this value is in $]0..3n]$.

4. A releases s bit by bit by opening h gradually as explained in Section 3. B checks each opening he receives and rejects if the check fails.

Note that in practice step 1 is only necessary once, and does not have to be repeated for every release. Nevertheless, we have included it here to make the formal proof easier.

Note also that all the actions in steps 2–3 can be parallelized, so that they take only three passes. The definition of p_k below is done with respect to this organization of the messages.

For this protocol we define the following, in order to be able to prove that it satisfies Definition 1:

- $P((m, n), s) = 1$ if and only if $s \in]0..3n]$ and $s^2 \bmod n = m$. Note that this predicate allows more than one possible s given m, n. This is no problem, however, because there are only three possible solutions for s given (n, m), and, from the first i bits of one solution, it is easy to compute the first i bits of any other solution.
- $p_k(i) = a(k) + 3 + i$, where $a(k)$ is the number of passes needed to execute step 1.
- The h_k^i functions are defined as follows: if the input view is shorter than $p_k(i)$ passes, then output i zeros. Else output the i bits opened by A in the final i passes of the input view.

We then have:

Theorem 1. *Under the factoring assumption, (A, B) is a secure release protocol with respect to the P, p_k, and h_k^i functions defined above.*

Proof. The first property is trivial by inspection of the protocol.

The proof of the second property is by contradiction—we make the basic assumption that property 2 of Definition 1 is not satisfied: there is an A, a constant c, and t's of infinitely many lengths, such that there are inputs s, r_A, k_B that make the probability in condition 2 of Definition 1 larger than k^{-c} for some $i = 1..k$. Recall that the common input t in this case is in fact a pair n, m. What this assumption means informally is that A has some strategy which with nonnegligible probability enables him to send i incorrect bits (i.e., bits that are not part of the signature on m) simultaneously with B accepting these bits as being genuine.

Let k be any input length for which the above holds, and assume that we are given a k-bit Blum integer N chosen with the same distribution B would have used. We now describe a poly-time nonuniform algorithm which factors N with probability at least a polynomial fraction, thus establishing a contradiction with the factoring assumption. The algorithm is nonuniform because we have to use the A and its inputs as given by the assumption.

We first choose a random element x modulo N, square it, and call the result g. We start up A with the inputs given by the assumption, and generate a random view of steps 1 and 2.

To this end, we send N, g to A and simulate the proof of knowledge of the factorization of N and the proof that N is a Blum integer with A acting as the verifier. Since the proofs are almost perfect zero-knowledge, A's behavior in what follows will have the same distribution as in "real life," except for a negligible amount of probability mass. The proof that g is a square we can do according to the protocol, as we know a square root x. Note that since this proof is perfect zero-knowledge, it is in particular witness-indistinguishable, so since we do not use x in the following, any root of g that can later be derived from messages sent by A is independent of x, and so leads to factorization of N with probability $1/2$.

A view of steps 1–2 is called *good* (that is, good for a cheating A) if it can be completed up to pass $p_k(i)$ with probability at least $k^{-c}/2$, and the h_k^i-value that can be computed from the completed view is incorrect. The assumption implies that the probability of $(A, \overline{B})$ completing pass $p_k(i)$ with an incorrect h_k^i-value is at least k^{-c}. This means that a random view of steps 1–2 is good with probability at least $k^{-c}/2$.

We now show how to factor N with probability at least $1/2$ minus a superpolynomially small fraction, assuming that the view of steps 1–2 we just created is good. By the above, this will be sufficient.

We repeatedly rewind A to the start of step 3 and issue randomly chosen requests in step 2 of the subprotocols. CHECK COMMITMENT and COMPARE COMMITMENTS. In doing so, we try to find correct answers to both requests in the same instance of the CHECK COMMITMENT (resp. the COMPARE COMMITMENTS) protocol. Since the probability of acceptance is at least $k^{-c}/2$, we can do this in polynomial time and succeed with probability essentially 1. By Lemmas 2 and 4, this tells us how to open h and v with the same value s, and how to open w with some value d, where $s \in]0 \cdots 3n]$ and $d \in]-2n - 1 \cdots 7n - 1]$. This means that we can write v as a base-g commitment to s^2, which is a legal way of opening v, since $s^2 < 2^{l-1}$. This in turn implies that we know how to open legally the number $g^m w^n v^{-1} \bmod N$ as a base-g commitment, namely, as $m + dn - s^2$. Recall, however, that we have at this point a view of steps 1–2, which is good and therefore does not lead to rejection by B. In particular the view of step 2 must include a correct opening by A of the same number $g^m w^n v^{-1} \bmod N$ to reveal 0. Hence, by Lemma 1, we get a factorization of N with probability $1/2$ unless $m + dn - s^2 = 0$ or, in other words, unless s is correct, i.e., is indeed a Rabin signature on m.

Hence if the s we have found is incorrect, our algorithm finds the factorization (with large probability) and halts. If it is correct, we execute the next and final part:

We use rewinding A to its state at the start of step 3, to generate random views of both step 3 and the following step 4, until we find one where pass $p_k(i)$ is completed, and the bits released by A are incorrect, i.e., inconsistent with the s we already know. Once again, since we start with a good view of steps 1–2, this can be done in polynomial time to succeed with probability essentially 1. However, this means that we have two different ways of opening part of the contents of h, which by the discussion following Lemma 1 gives us a factoriza-

tion of N with probability $1/2$. This concludes the proof of Property 2 of Definition 1.

Property 3 is proved by first observing that step 1 contains a proof of knowledge of the factorization of N. Thus if B completes step 1 with probability more than a polynomial fraction, we can always find the factorization. Moreover, B cannot get a nonsquare accepted as g with probability more than 2^{-k}. Thus we see that, except for negligibly few cases, we can simulate the conversation perfectly by sending random squares in place of all commitments, and opening them as needed using our knowledge of the factorization of N. In particular, step 3 is simulated using Lemmas 3 and 5, and step 4 is simulated using the input we are given, which tells us what the least-significant i bits of s are. □

It is easy to see that this protocol can be modified to release, for example, an RSA signature with public exponent 3 by introducing a new commitment u, such that $h = BC_g(R, s)$, $v = BC_h(R', s)$, and $u = BC_v(R'', s)$, which will make u a base-g commitment to s^3. It is also clear, however, that this quickly becomes impractical with increasing public exponents.

6. Release of El Gamal Signatures

In this section we sketch how to release El Gamal signatures. We first recall the usual setup of the El Gamal signature scheme: a large prime p is chosen, together with a generator α of Z_p^*. A private key x is a number in $[0..p-1[$, while the corresponding public key is $y = \alpha^x \bmod p$. Messages are numbers in $[0..p-1[$, and a signature on message m is a pair (r, s) such that

$$\alpha^m \equiv y^r \cdot r^s \bmod p.$$

For the owner of x, a signature is easy to compute by choosing a random w relatively prime to $p-1$, setting $r = \alpha^w \bmod p$ and solving the equation $m = (xr + ws) \bmod (p-1)$ for s, where $s \in [0..p-1[$. It is conjectured that computing signatures from scratch is as hard as finding x from y. In the following we assume that α really is a generator of Z_p^*. In practice, this assumption may be justified because p, α was generated by a trusted party, or because the factorization of $p-1$ is made public, which makes it easy to test α.

The following is based on the observation that the gradual release of a discrete log mod p is sufficient for the release of an El Gamal signature. The idea is that we first reveal r and then release s bit by bit. Note that s is the discrete log base r of $\beta = \alpha^m y^{-r} \bmod p$. This reduces the problem to that of proving that the discrete log base r of β equals the contents of a base-g commitment $h = BC_g(R, s)$ computed as in Section 3.

We assume that the prover (sender) A knows such a discrete log s in the interval $I =](p-1)..2(p-1)]$. Such an s can always be obtained from an El Gamal signature by adding $p-1$ to the last component.

Using a technique similar to that of COMPARE COMMITMENTS, we get the following protocol, which will be a proof that A knows a suitable s in $]0..3(p-1)]$. If A uses an s in I, the protocol will be zero-knowledge.

PROTOCOL TRANSFER DISCRETE LOG

This protocol is executed with $\beta = r^s \bmod p$ and commitment $h = BC_g(R, s)$ as common input. s is private input to A. It convinces B that A knows how to open h to reveal a value in $]0..3(p-1)]$, which is also a discrete log of β base r.

Execute the following k times in parallel:

1. A chooses t_1 uniformly in $]0..p-1]$, and sets $t_2 = t_1 - (p-1)$. He sends to B the unordered pair $((T_1, T_1'), (T_2, T_2'))$, where each component of the pair is ordered and is defined by $(T_i, T_i') = (BC_g(S_i, t_i), BC_g(S_i', z_i))$, where $z_i = r^{t_i} \bmod p$.
2. B requests to see one of the following:
   (a) Opening of (T_i, T_i') for both $i = 1$ and 2.
   (b) Opening of $h \cdot T_i \bmod N$ and T_i', where A chooses i such that $s + t_i \in I$.
3. In the first case of step 2, B checks that the number contained in T_i is the discrete log base r of the number contained in T_i', and that this discrete log is in $]-(p-1)..p-1]$. In the second case, B checks that the number contained in $h \cdot T_i \bmod N$ is the discrete log base r of $\beta z_i \bmod p$, and that the discrete log revealed is in I.

B outputs reject and stops if any of the openings are not correctly done, or if any of the checks required are not satisfied.

The following two lemmas give the basic properties of this protocol:

Lemma 6. *Given correct answers to both* (a) *and* (b) *in one instance of steps* 1–3 *above, either* (R, s) *such that* $s \in]0 \cdots 3(p-1)]$, $h = BC_g(R, s)$, *and* $\beta = r^s \bmod p$ *or a square root of* g *modulo* N *can be computed efficiently.*

Proof. Note that A must open T_i' in both case (a) and (b). If these openings are not consistent, we get a square root of g by Lemma 1. Otherwise, what we have from the correct answers is a 4-tuple of numbers (u, U, v, V) such that

$$T_i = BC_g(U, u) \quad \text{and} \quad h \cdot T_i = BC_g(V, v)$$

and a pair (z, Z) such that

$$T_i' = BC_g(Z, z) \quad \text{and} \quad z = r^u \bmod p, \qquad \beta z = r^v \bmod p.$$

Furthermore, $v \in I$ and $u \in]-(p-1)..(p-1)]$. From this it follows trivially that $h = BC_g(V/U, v-u)$ and that $\beta = r^{v-u} \bmod p$. □

Lemma 7. *Given the factorization of* N, *any* B*'s view of TRANSFER DISCRETE LOG when talking to* A *can be simulated perfectly, provided* $s \in I$.

Proof. Follows by trivial modifications of the proof of Lemma 3. □

To define the parameters of the release protocol, we define the shared input to A and B to be p, α, y, r, and m, all of length $|p|$, so that k, the length of this input, is $k = 5|p|$. The private input to A is a $|p|$-bit string s. The predicate P for this situation is defined such that $P(p, \alpha, y, r, m, s) = 1$ if and only if $\alpha^m = y^r r^s \bmod p$ and $s \in]0..3(p-1)]$.

Note that with this definition of the shared input, we have implicitly assumed that the sender will make r known immediately at the start of the protocol. This does not lead to a security problem, because the receiver could by himself easily simulate such an r by first finding a k such that $(k, p-1) = 1$ and setting $r = \alpha^k$. For any such r there is an $s \in I$ such that (r, s) signs m. In other words, seeing r in the beginning does not help B to compute the signature ahead of time.

The following gives a brief outline of the complete release protocol for El Gamal signatures:

PROTOCOL RELEASE EL-GAMAL SIGNATURE
This protocol is executed with p, α, y, r, and m as common input to A and B. Private input to A is s, such that $\alpha^m = y^r r^s \bmod p$, and $s \in](p-1)..2(p-1)]$. The protocol convinces B that he will in the last step receive bits of $s \in]0..3(p-1)]$, such that the pair (r, s) is an El Gamal signature on m using public key y.

1. B chooses the parameters of the bit commitment scheme N, g. These are verified interactively as explained in Section 3.
2. A sends $h = BC_g(R, s)$ to B.
3. A uses the TRANSFER DISCRETE LOG protocol to prove that the value s contained in h also satisfies that $r \in]0..3(p-1)]$ and $r^s \bmod p = \beta$, where $\beta = \alpha^m \cdot y^{-r}$.
4. A releases s bit by bit by opening h gradually as explained in Section 3. B checks each opening he receives and rejects if the checks fail.

The h_k^i and the p_k functions for this protocol are defined similarly to those in the previous section.

Theorem 2. *Under the factoring assumption, the protocol outlined above is a secure release protocol with respect to the P, h_k^i, and p_k functions defined in this section.*

Proof. The first property is trivial. The second one is proved in essentially the same way as in Theorem 1: since the TRANSFER DISCRETE LOG protocol is a proof of knowledge, we can use rewinding of A to compute an s that both opens h and satisfies $r^s \bmod p = \beta$. Thus s is by definition the correct secret. Therefore a view of the protocol that leads to an incorrect value must give us a way of opening h that is inconsistent with s, and therefore enables us to compute a root of g, and factor N with large probability. The third property

follows easily from Lemma 7 and the fact that B is required to give a proof of knowledge of the factorization of N. □

We remark that the same basic idea can also be used for the release of signatures in other discrete-log-based schemes such as NIST DSA and Schnorr's signature scheme. This is because these signatures, like El Gamal signatures, include a discrete logarithm that is hard to compute without knowledge of the secret key. Thus the sender can reveal all components of the signature except this discrete log, and release this gradually using the above methods.

Acknowledgments

The author would like to thank the anonymous referees for many constructive suggestions which greatly improved the presentation.

Appendix. Formalizing Fairness of Induced Exchange Protocols

In this appendix we give one possible way of formalizing fairness of an induced exchange protocol, and uniform hardness of a predicate. We then show that an exchange protocol induced from a secure release protocol for a uniformly hard predicate will be fair.

The formalization we give here is certainly not the only one possible. In particular, the conditions in the definition of uniform hardness have been made quite demanding, to avoid technical complications in the following arguments.

For simplicity, we restrict ourselves to predicates P which define unique secrets, i.e., for each t, there is at most one s such that $P(t, s) = 1$.

The notation from Section 2 is used without further introduction. We also need some new notation: let f be a function from the natural numbers to the reals, let K be an infinite set of natural numbers, and let c be a real constant. Then

$$f(k) \simeq c \qquad \text{for} \quad k \in K$$

should be taken to mean that

$$\text{for any polynomial } p, \qquad |f(k) - c| \leq 1/p(k) \quad \text{for all large enough } k.$$

When defining fairness of an induced exchange protocol (X, Y), we are trying to capture the intuition that if some (possibly cheating) $\tilde{Y}$ is capable of computing X's secret after release of, say $q(k)$ bits, then X should also be able to compute Y's secret. This leads to the following:

Definition A.1. Let (X, Y) be the exchange protocol induced from release protocol (A, B) for predicate P. Then (X, Y) is said to be *fair*, if, for each probabilistic polynomial time $\tilde{Y}$, there is a probabilistic polynomial-time algo-

rithm $A(\tilde{Y})$, which gets a view of X as input and tries to compute s_Y. The algorithm $A(\tilde{Y})$ should satisfy the following:

For each function q with $0 \leq q(k) \leq k$, we have that

if $Pr(\tilde{Y}, q(k)) \geq 1/p(k)$ for a polynomial p and k in some infinite set K
then $Pr(A(\tilde{Y}), q(k)) \simeq 0$ for $k \in K$

Here, $Pr(\tilde{Y}, q(k))$ is the probability that $\tilde{Y}$ stops after release of at most $q(k)$ bits of s_X, and outputs the correct s_X; this probability is taken over the choice of t_X, t_Y and the coinflips of X and $\tilde{Y}$. Furthermore, $Pr(A(\tilde{Y}), q(k))$ is the probability that $A(\tilde{Y})$ fails to extract the correct s_Y from the input view, *and* that at least $q(k)$ bits of s_X are released in the view; this probability is taken over the choice of t_X, t_Y and the coinflips of X, $\tilde{Y}$, and $A(\tilde{Y})$.

A symmetric condition similar to the above should hold for all probabilistic polynomial time $\tilde{X}$.

We now define what it means for a predicate P to be uniformly hard. Recall that such hardness intuitively means that the problem of computing from t and some bits of s (where $P(t, s) = 1$) all the bits of s is of the same difficulty for nearly all instances of the same size. We formalize this by saying that if you have enough information to solve the problem for a nonnegligible fraction of the instances, then you can in fact solve nearly all instances.

Definition A.2. Let P be a predicate on two variables with properties as required for a release protocol (see Section 2.1). Let $\mathscr{A}$ be a probabilistic polynomial-time algorithm which gets as input a string t and $s|_q$ (q bits of s), where $P(t, s) = 1$; the algorithm tries to produce s as output.

The predicate P is said to be *uniformly hard* if, for each k, there is a distribution π_k on k-bit strings, such that:

- In probabilistic polynomial time, one can sample a pair (t, s) such that $P(t, s) = 1$ and t is distributed according to π_k.
- For any $\mathscr{A}$ as above, there is another probabilistic polynomial-time algorithm $\mathscr{A}_0$ with input and output as for $\mathscr{A}$. Algorithm $\mathscr{A}_0$ satisfies that, for any function q with $0 \leq q(k) \leq f(k)$,

  if $Prob(\mathscr{A}(t, s|_{q(k)}) = s) \geq 1/p(k)$ for a polynomial p and k in some infinite set K
  then $Prob(\mathscr{A}_0(t, s|_{q(k)}) = s) \simeq 1$ for $k \in K$

  These probabilities are taken over the choice of (t, s) and the coinflips of $\mathscr{A}$ (resp. $\mathscr{A}_0$).

We are now ready to state the result of this appendix:

Theorem A.1. *Let (A, B) be a secure release protocol for the uniformly hard predicate P. Then the exchange protocol (X, Y) induced from (A, B) is fair.*

Proof. Let q be a function satisfying the if-clause of Definition A.1. Note that we can use the machine $\tilde{Y}$ to build an algorithm $\mathscr{A}$ that complies with Definition A.2. This algorithm takes as input t of length k and $s \mid_{q(k)}$ (where $P(t, s) = 1$) and works as follows:

1. Sample a pair (t_Y, s_Y) according to Definition A.2. Give s_Y as private input to $\tilde{Y}$ and use t, t_Y as the common inputs—thus t plays the role as t_X.
2. Using the simulator guaranteed by Definition 1, simulate the exchange of the first q bits of s and s_Y taking place between X and $\tilde{Y}$.
3. If $\tilde{Y}$ stops before the $(q + 1)$st bit of s is required and outputs the correct s, then we copy s to the output and stop. Otherwise output a random string and stop.

Since the simulation in step 2 is statistically close to a real conversation, it is clear that when $k \in K$, $Pr(\tilde{Y}, q(k))$ is equal to $Prob(\mathscr{A}(t, s \mid_{q(k)}) = s)$ except for a superpolynomially small amount. Hence, if $q(k)$ satisfies the if-clause of Definition A.1, algorithm $\mathscr{A}$ just described satisfies the if-clause of Definition A.2. Therefore, we may assume that we have an algorithm $\mathscr{A}_0$ satisfying that $Prob(\mathscr{A}_0(t, s \mid_{q(k)}) = s) \simeq 1$ for $k \in K$.

We can now build algorithm $A(\tilde{Y})$ required by Definition A.1. We have as input X's view of a conversation with $\tilde{Y}$. We have access to the machine $\tilde{Y}$, but of course not to the inputs used by $\tilde{Y}$ when the input view was generated. The algorithm works as follows:

1. Extract from the input view $q(k) - 1$ bits released by $\tilde{Y}$. Call the result of this y. If less than $q(k) - 1$ bits were released, we fail and stop.
2. Run $\mathscr{A}_0$ twice, on inputs $(t_Y, 1\|y)$ and $(t_Y, 0\|y)$ (here, $\|$ means concatenation).
3. If one of the two runs of $\mathscr{A}_0$ gives the correct s_Y, output this, else output something random.

To see that this algorithm succeeds with the required probability, note that the structure of the exchange protocol forces $\tilde{Y}$ to release at least $q(k) - 1$ bits of its secret (and have these accepted by X) whenever $q(k)$ or more bits of s_X are released. Moreover, Definition 1 guarantees that in nearly all cases where these bits are accepted by X, they are in fact correct bits of s_Y. However, now the properties of $\mathscr{A}_0$ guarantee that in nearly all cases where we get $q(k) - 1$ correct bits of s_Y, we will obtain the correct s_Y in step 2. □

References

[1] W. Alexi, B. Chor, O. Goldreich, and C. P. Schnorr. RSA and Rabin functions: certain parts are as hard as the whole, *Proc. 25th FOCS*, 1984, pp. 449–457.

[2] M. Ben-Or, O. Goldreich, S. Micali, and R. Rivest. A fair protocol for signing contracts, *IEEE Trans. Inform. Theory*, vol. 36, 1990, pp. 40–46.

[3] M. Blum. Three Applications of the Oblivious Transfer, Dept. of EECS, University of California, Berkeley, 1981.

[4] M. Blum. Coin-flipping by telephone, *Proc. IEEE Spring COMPCOM*, 1982.

[5] M. Blum. How to exchange (secret) keys, *ACM Trans. Comput. Systems*, vol. 1, 1983, pp. 175–193.

[6] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge, *J. Comput. System Sci.*, vol. 37, 1988, pp. 156–189.
[7] E. F. Brickell, D. Chaum, I. Damgård, and J. van de Graaf. Gradual and verifiable release of a secret, *Proc. Crypto 87*, Lecture Notes in Computer Science, vol. 293, Springer-Verlag, Berlin, 1988, pp. 156–166.
[8] J. Cleve. Controlled gradual disclosure schemes for random bits and their applications, *Proc. Crypto 89*, Lecture Notes in Computer Science, vol. 435, Springer-Verlag, Berlin, 1990, pp. 573–588.
[9] I. Damgård. Collision free hash functions and public key signature schemes, *Proc. EuroCrypt 87*, Lecture Notes in Computer Science, vol. 304, Springer-Verlag, Berlin, 1988, pp. 147–158.
[10] I. Damgård. Practical and provably secure release of a secret and exchange of signatures, *Proc. EuroCrypt 93*, Lecture Notes in Computer Science, vol. 765, Springer-Verlag, Berlin, 1994, pp. 200–217.
[11] S. Even, O. Goldreich, and Z. Lempel. A randomized protocol for signing contracts, *Proc. Crypto 82*, Plenum, New York, 1983, pp. 205–210.
[12] S. Even and Y. Jacobi. Relations Among Public Key Signature Systems, Comput. Sci. Dept., Technion, Haifa, March 1980.
[13] U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity, *J. Cryptology*, vol. 1, no. 2, 1988, pp. 77–94.
[14] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design, *Proc. 27th FOCS*, 1986, pp. 174–187.
[15] S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority, *Proc. Crypto 90*, Lecture Notes in Computer Science, vol. 537, Springer-Verlag, Berlin, 1991, pp. 77–93.
[16] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems, *SIAM J. Comput.*, vol. 18, 1989, pp. 186–208.
[17] J. van de Graaf and R. Peralta. A simple and secure way to show the validity of your public key, *Proc. Crypto 87*, Lecture Notes in Computer Science, vol. 293, Springer-Verlag, Berlin, 1988, pp. 128–134.
[18] J. Håstad and A. Shamir. The cryptographic security of truncated linearly related variables, *Proc. ACM Symp. on Theory of Computing*, 1983, pp. 356–362.
[19] R. Impagliazzo and M. Yung. Direct minimum knowledge computations, *Proc. Crypto 87*, Lecture Notes in Computer Science, vol. 293, Springer-Verlag, Berlin, 1988, pp. 40–51.
[20] M. Luby, S. Micali, and C. Rackoff. How to simultaneously exchange a secret bit by flipping a symmetrically-biased coin, *Proc. 24th FOCS*, 1983, pp. 11–22.
[21] M. Rabin. How to exchange secrets by oblivious transfer, Tech. Memo TR-81, Aiken Comput. Lab., Harward University, 1981.
[22] T. Tedrick. Fair exchange of secrets, *Proc. Crypto 84*, Lecture Notes in Computer Science, vol. 196, Springer-Verlag, Berlin, 1985, pp. 434–438.
[23] M. Tompa and H. Woll. Random self-reducibility and zero-knowledge proofs of information possession, *Proc. 28th FOCS*, 1987, pp. 472–482.
[24] U. Vazirani and V. Vazirani. Trapdoor pseudorandom number generators with applications to cryptographic protocol design, *Proc. 24th FOCS*, 1983, pp. 23–30.
[25] M. Waidner and B. Pfitzmann. The dining cryptographers at the disco: unconditional sender and recipient untraceability with computational secure serviceability, *Proc. EuroCrypt 89*, Lecture Notes in Computer Science, vol. 434, Springer-Verlag, Berlin, 1989, p. 690.
[26] H. C. Williams. A modification of the RSA public key cryptosystem, *IEEE Trans. Inform. Theory*, vol. 26, 1980, pp. 417–426.
[27] A. C. Yao. How to generate and exchange secrets, *Proc. 27th FOCS*, 1986, pp. 162–167.