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

Ivan Bjerre Damgård

Aarhus University, Mathematical Institute

Abstract. We present a protocol that allows a sender to gradually and verifiably release a secret to a receiver. We argue that the protocol can be efficiently applied to exchange secrets in many cases, for example 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 3-pass initial phase, after which each bit (or block of bits) of the signature can be released non-interactively (i.e. by sending 1 message). The necessary computations can be done in a few seconds 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.
We also point out a simple method by which any type of signatures can be applied to fair contract signing using only one signature.

1 Introduction

1.1 The Basic Problem

Suppose parties A and B each possess a secret, s_A and s_B resp. 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. But if the parties do not trust each other, it is clear that none of them are 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'll 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, is not attractive and probably not very realistic either.

Thus we can distill a basic primitive (introduced in [4]) which we will 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 and 2.3). 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

Exchange and release of secrets has attracted a lot of attention in the past, and a large body of literature exists on the subject [4, 5, 6, 7, 9, 10, 16, 17, 18, 19, 20, 22, 24]. However, since the discovery of zero-knowledge proofs and arguments for any NP-language [3, 12], the problem has lost most of its theoretical interest 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 1 bit at a time, see for example [18, 13]. It is even possible to make the choice of bits to release adaptive [17].

The resulting solutions would be 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 [4], a factorization in [6]. 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. [4], 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 (but such that the commitment has constant length) and can be opened bit by bit. 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 will discuss to what extent such a fair exchange follows from a gradual and verifiable release.

But 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 exchange gradually 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, Goldreich, Micali and Rivest show how to avoid this problem, if we assume that a judge is available to settle disputes. Moreover, their protocol can make use of any signature scheme. The difference to our work is first the assumption about the judge, and secondly that the protocol of [2] is a dedicated protocol for solving the contract signing problem: it does not implement an exchange or a relase of secrets.

The protocol in [2] involves a certain computational overhead: the signature scheme must be employed a large humber of times by both parties. In Section 7 we show that this overhead can be avoided.

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. In subsection 2.3 we argue that the model is in fact useful in many practical situations.

2.1 Release Protocols

In this section we give a formal definition of a secure release protocol. Intuitively, we are modelling 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 not more, i.e. he should be in the same situation as if he had been given t and the bits of s by an oracle.

We will think of the pair (A, B) as interactive Turing machines as defined in [11]. In particular, both A and B will be polynomial time bounded in the input length, and are equipped with knowledge tapes containing their private inputs. In the following X will mean A or B. Following [11], 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. X will represent $\bar{X}$ or $\tilde{X}$.

The properties of (A, B) will be defined with respect to a fixed polynomial time computable predicate P. P takes as input a k-bit string t and a bit string s of length at most $f(k)$, where f is a polynomial.

A receives as private input on its knowledge tape a string s of length at most $f(k)$, while the k-bit string t is common input to A and B. B receives the string k_B on its knowledge tape. $s|_i$ denotes the first i bits of s ($s|_0$ is the empty string). The interesting case is of course when $P(t, s) = 1$.

The event that one party sends a message to the other is called a *pass*. Passes are numbered ordinarily, starting from 1. The protocol is required to define a series of increasing functions $\{p_k\}_{k=1}^{\infty}$, where

$$p_k : \{0...f(k)\} \rightarrow \mathbf{N},$$

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 B is able to compute $s|_i$. After each pass, the participant receiving a message may output "reject" and stop, indicating that cheating has been detected. We say that (A, B) *completes pass* i if no party outputs reject after pass i.

As usual, 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 contents of the random tape of party X. $View^i_X(t, s, r_A, k_B, r_B)$ denotes party X's view of the truncated protocol where we only consider passes number 1 through i (note that this view may be shorter than i passes if the protocol stops earlier). In the following, $View_{\tilde{B}}(...)$ will always refer to a conversation with A, while $View_B(...)$ will refer to a conversation with $\bar{A}$.

Finally, the protocol must define a set of polynomial time computable functions $\{h^i_k | \ k = 1..\infty, i = 1..f(k)\}$, such that h^i_k takes as input a sample of $View^{p_k(i)}_B(t, s, r_A, k_B, r_B)$. 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. An h^i_k-value 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:

- h_k^i-values computed on views of conversations between $\bar{A}$ and $\bar{B}$ are always correct.
- $\forall A \forall c \exists k_0 \forall |t| > k_0 \forall s, r_A, k_B \forall i = 1..f(k)$:

  $$Prob(h_k^i(View_{\tilde{B}}^{p_k(i)}(t, s, r_A, k_B, r_B)) \text{ incorr. and } (A, \tilde{B}) \text{ completes pass } p_k(i))$$
  $$\leq k^{-c}$$

  The probability is taken over the choice of r_B.
- For each B, and for each $i = 0..f(k)$, there exists an expected polynomial time machine M_B^i, which on input bit strings x, t, k_B and with random tape r_M simulates B's view of the first $p_k(i)$ passes of the conversation with $\bar{A}$. Let $M_B^i(x, t, k_B, r_M)$ denote M_B^i's output, considered as a random variable with distribution taken over r_M.
  We then require that for $i = 0...f(k)$, whenever $x = s|_i$ for some s with $P(t, s) = 1$, then the distribution of $M_B^i(x, t, k_B, r_M)$ 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, and where s' is any string such that $s'|_i = s|_i$ and $P(t, s') = 1$.

Remark

- 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 a simulator for each $i = 0..k$, because we want A to be protected, even if $\tilde{B}$ stops before all bits are released. The best we can do in such a case is to require that $\tilde{B}$ 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 shall discuss to what extent release protocols can be used to build fair exchange protocols. It is clear that if parties X and Y possess secrets s_X, s_Y, resp., 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 [24]: 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 uniquely determined 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 half way 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. We therefore have to restrict to a set of "nicer" cases, where it is possible to connect the two problems. A first step in this direction is to require that s_X, s_Y are defined by the same predicate (i.e. $P = P'$), and that the corresponding public strings t_X, t_Y are 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 non-negligible fraction of the t's, finding 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 non-negligble 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 non-negligible 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. It follows from Definition 1 that from this information only, the views of X, resp. Y can be simulatewd. So except for perhaps 1 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, one can formalize this reasoning. In this paper, however, where we focus on practical protocols, we leave this to the reader. In stead we concentrate on the question whether the types of secrets one might want to exchange in practice are likely to have a uniform hardness property 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 poynomial 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.

Since El Gamal signatures are known to satisfy a similar property, we 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 are too demanding in practice, in particular the assumption that t_X and t_Y are identically distributed. What if Y could somehow manipulate the distribution of t_X and/or t_Y, presumably to make life easier for himself? However, if messages are hashed before they are signed - as is nearly always the case in practice - he is not likely to benefit from this: 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, make public some information connecting w and the actucal secrets, and then release w bit by bit, see e.g. [24]. Thus a bit-by-bit release as defined here can also be useful as a building block in other protocols.

3 A Bit Commitment Scheme

In this section, we define the bit commitment scheme we will use, and prove its basic properties. To set up the commitment scheme, B must generate and make public a k-bit Blum integer N, and g, a random quadratic residue modulo N. Also, B must in zero-knowledge prove that he knows the two prime factors of N [21], prove that they are both congruent to 3 modulo 4 [1] [15], and that g is a quadratic residue [14]. The methods for doing this are well known and quite efficient, and in any case this step will only be necessary once, at system start-up time.

In this 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. A can now 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

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

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 [8]. 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 [8]. The same type of function was used in [23] for the purpose of fail-stop signatures.

The basic properties of this commitment scheme are established in the following lemma:

Lemma 1 $BC_g(R, s)$ has distribution independent of s, when R is a uniformly chosen square mod N.
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

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

[1] Actually, [15] only proves that $N = p^r q^s$, where r, s are odd and p, q are 3 modulo 4. But even for such numbers, squaring is a permutation of the quadratic residues, and this is the property we need.

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 [8]. 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

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

and

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

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 1 bit can be easily 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 will need the following assumption on hardness of factoring:

Factoring Assumption There exists a probabilistic polynomial time algorithm Δ which on input 1^k outputs a k-bit Blum integer N, such that for any probabilistic polynomial size circuit families 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.

It should be noted that from a practical point of view, this assumption is actually stronger than necessary for our protocol. What *is* needed is that the sender A cannot factor N before the protocol halts. Even a polynomial time factoring algorithm may not help him to do this.

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 will therefore need the following protocol, which is based on the proof system from [4], 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 $I =]a...b]$ and put $e = b - a$. We define $I \pm e =]a-e...b+e]$. These parameters must be chosen such that $I \pm e$ is contained in the legal range for openings of commitments $]-2^{l-1}...2^{l-1}[$. 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
Execute the following k times in parallel:

1. A chooses t_1 uniformly in $]0..e]$, and puts $t_2 = t_1 - e$. 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 $c \cdot T_i \bmod N$, 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:

Lemma 2 Given correct answers to both a) and b) in one instance of steps 1-3 above, one can efficiently compute a pair R, s such that $s \in I \pm e$ and $c = BC_g(R, s)$.

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

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

where $x \in]-e..e]$, $y \in I$. 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 Given 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, independently 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
Execute the following k times in parallel:

1. A chooses t_1 uniformly in $]0..e]$, and puts $t_2 = t_1 - e$. 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:

Lemma 4 Given correct answers to both a) and b) in one instance of steps 1-3 above, one can efficiently compute R, R', s such that $s \in I \pm e$ and $c = BC_g(R, s)$, $c' = BC_h(R', s)$.

Proof Trivial from the proof of Lemma 2□

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.

Proof Trivial from the proof of Lemma 3□

5 Release of Rabin and RSA Signatures

We are now ready to present a complete protocol for release of a Rabin signature. The common input to the parties will be a modulus n and a message $m \in]0...n[$, while A's private input will be a number s in $]n..2n]$ such that $s^2 \bmod n = m$. Thus k will be $2|n|$, where $|n|$ is the bit length of n. For all commitments in the following, we will use $l = 2|n| + 3$. The protocol has the following steps:

PROTOCOL RELEASE RABIN SIGNATURE

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 $I =]n-1...4n-1]$ to prove that he knows how to open w to reveal a value in $]-2n-1..7n-1]$.
   A uses the COMPARE COMMITMENTS protocol with $I =]n...2n]$ to prove that he knows how to open h and v to reveal the same value, 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 Step 2-3 can be parallelized, so that they take only 3 passes. The definition of p_k below will be done with respect to this organization of the messages.

For this protocol, we define $P_k(m, n, s) = 1$ 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 3 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.

Assume step 1 takes $a(k)$ passes. Then we define $p_k(i) = a(k) + 3 + i$.

The h_k^i functions are defined as follows: if the input view is shorter than $p_k(i)$ passes, then output i 0's. 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. So assume that there exists 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 of the definition be larger than k^{-c} for some $i = 1..k$.

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 non-uniform algorithm which factors N with probability at least a polynomial fraction, thus establishing a contradiction with the factoring 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 the sequel 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 will not use x in the sequel, 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 Step 1-2 is called *good*, 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 $(A, \bar{B})$ completing pass $p_k(i)$ with an incorrect h_k^i-value is at least k^{-c}. This means that a random view of Step 1-2 is good with probability at least $k^{-c}/2$.

Below we show how to factor N with probability at least 1/2 minus a super-polynomially small fraction, assuming that the view of Step 1 we just created is good. By the above, this will be sufficient.

By rewinding A to the start of Step 3 and issuing randomly chosen requests, 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 $s \in]0...3n]$ and $d \in]-2n-1...7n-1]$. 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 legally open the number $g^m w^n v^{-1} \bmod N$ as a base-g commitment, namely as $m + dn - s^2$. Since A has just told us how to open the same number as a 0, we get a factorization of N by Lemma 1 with probability 1/2 unless $s^2 = m + dn$, in other words, unless s is indeed a Rabin signature on m.

We now use rewinding A to its state at the start of Step 3, to generate random views of the conversation, until we find one where pass $p_k(i)$ is completed, and the bits released by A are incorrect. Once again, by the assumption, this can be done in polynomial time to succeed with probability essentially 1. But this means that we have two different ways of opening part of the contents of h, which by Lemma 1 gives us a factorization of N with probability 1/2.

The third condition in Definition 1 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 non-square 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 Lemma 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$\square$

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 k-bit 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 k relatively prime to $p-1$, putting $r = \alpha^k \bmod p$ and solving the equation $m = xr + ks \bmod p-1$ for s. It is conjectured that computing signatures from scratch is a as hard as finding x from y. In the following, we assume that α really is a generator of Z_p^*. In practice, this 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 gradual release of a discrete log mod p is sufficient for relase of an El Gamal signature. The idea is that we first reveal r and then release bit by bit s, which will be 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 will 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
Execute the following k times in parallel:

1. A chooses t_1 uniformly in $]0..p-1]$, and puts $t_2 = t_1 - (p-1)$. 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 $(T_i, T_i') = (BC_g(S_i, t_i), BC_g(S_i', r^{t_i} \bmod p))$.
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 $\beta z \bmod p$ where z is the number contained in T_i', 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, one can efficiently compute either R, s such that $s \in]0...3(p-1)]$ and $h = BC_g(R, s)$, $\beta = r^s \bmod p$; or a square root of g modulo N.

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 numbers u, U, v, V such that

$$T_i = BC_g(U, u) \quad h \cdot T_i = BC_g(V, v)$$

and z, Z such that

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

Furthermore, $v \in I$ and $u \in]-(p-1)..(p-1)]$. From this 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 $\bar{A}$ 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 put $k = |p| + 1$, the bit length of p plus 1, and we define the shared input to A and B to be p, α, y, r and m, all of length $|p|$. The private input to A is a k 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 implicitely 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 easily by himself simulate such an r by first finding a k such that $(k, p-1) = 1$ and putting $r = \alpha^k$. 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.

In the complete release protocol, B will set up the bit commitment scheme, A will commit to s by sending h as computed above, use TRANSFER DISCRETE LOG to show that the commitment really contains the discrete log base r of β, and will finally release s bit by bit as explained in Section 3.

The h_k^i and the p_k functions are defined similarly to what was done 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 Sketch The first property is trivial. The second one is proved in essentially the same way as for 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 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.

7 Efficient Contract Signing

As explained in the introduction, one possible application of exchange of signatures is to fair contract signing.

However, under the assumption that intervention by a third party is possible, a different solution to contract signing was proposed in [2]. As pointed out there, that solution will sometimes be superior to simply exchanging signatures because it will work, even if one party has much more computing power than the other, and because any signature scheme can be used.

Very briefly, the solution works by having the parties first sign a message stating that they intend to use the protocol below to sign the contract C. We call this message $M(C)$. Signing $M(C)$ commits neither party to C, but prevents them from claiming a different contents of C later.

They then exchange signatures on messages of the form "this signature on contract C should be considered valid with probability p". This exchange is repeated with increasing values of p. When p reaches 1, the attached signature can be considered an ordinary signature on C. But if for example A stops early, B can appeal to a jugde, showing him the last signed message he received from A. Let p_A be the p-value used in this message. The judge then takes a biased random decision: if the signature is valid, then with probability p_A, he decides that the contract is binding for party A. This introduces some computational overhead, compared to simply signing the contract: a new signature is necessary each time we increase the p-value. Moreover, the number of signatures needed increases with the "granularity" with which the contract signing takes place.

The purpose of this section is to point out that all but one of these signatures can be replaced by simple computations of a one-way function. With the techniques known in practice today, this will much more efficient.

Our method makes use of an arbitrary one-way function f. Such a function always exists, if digital signatures do. Moreover, in practice, we have good candidates for one-way functions based on conventional cryptography that are much more efficient to compute than for example an RSA signature.

The idea now is to let A and B initially each choose a list of f-inputs, $a_1, ..., a_t$, resp. $b_1, ..., b_t$. They then exchange the f-values $f(a_i), f(b_i)$ for $i = 1..t$, and all these values are included in $M(C)$, which is signed initially by both parties.

In stead of exchanging signed messages with increasing p-values, the parties now exchange the f-preimages they have chosen, i.e. A starts by sending a_1, waits to receive b_1, if b_1 is valid he then sends a_2, etc. The only other change needed in [2] is in the procedure of the judge: we fix a rule, stating what the bias of his decision should be, as a function of how many valid preimages the complaining party can present to him.

Since as mentioned, almost all known digital signature schemes are much slower in practice than computing a conventional one-way function (such as MD4 for example), this protocol requires very little extra computational effort compared to simply signing the contract without being concerned about fairness.

References

1. Alexi,W., Chor, B., Goldreich, O. and Schnorr, C.P.: "RSA and Rabin Functions: Certain Parts Are as Hard as the Whole". Proc. of the 25th FOCS, 1984, pp. 449-457.
2. Ben-Or, Goldreich, Micali and Rivest: *A Fair Protocol for Signing Contracts*, IEEE Trans. Info. Theory, Vol.36, 1990, pp.40-46.

3. G.Brassard, D.Chaum and C.Crépeau: *Minimum Disclosure Proofs of Knowledge*, JCSS.
4. Brickell, Chaum, Damgård and van de Graaf: *Gradual and Verifiable Release of a Secret*, Proc. of Crypto 87, Lecture Notes in Computer Science, Springer Verlag.
5. Blum: *Three Applications of the Oblivious Transfer*, Dept. of EECS, University of California, Berkely, 1981.
6. Blum: *How to Exchange (Secret) Keys*, ACM Transactions on Computer Systems, vol.1, 1983, pp.175-193.
7. Cleve: *Controlled Gradual Disclosure Schemes for Random Bits and Their Applications*, Proc. of Crypto 89, Lecture Notes in Computer Science, Springer Verlag.
8. I.Damgård: *Collision Free Hash Functions and Public Key Signature Schemes*, Proc. of EuroCrypt 87, Lecture Notes in Computer Science, Springer Verlag.
9. Even, Goldreich and Lempel: *A Randomized Protocol for Signing Contracts*, Proceedings of Crypto 82, Plenum Press.
10. Even and Jacobi: *Relations Among Public Key Signature Systems*, Comp. Sci. Dept., Technion, Haifa Israel, March 1980.
11. U.Feige, A.Fiat and A.Shamir: *Zero-Knowledge Proofs of Identity*, J.Crypt. Vol1, no.2, 1988.
12. O.Goldreich, S.Micali and A.Wigderson: *Proof that Yield Nothing but their Validity and a Methodology of Cryptographic Protocol Design*, Proc. of FOCS 86.
13. S.Goldwasser and L.Levin: *Fair Computation of General Functions in Presence of Immoral Majority*, Proc. of Crypto 90, Spinger Verlag LNCS series.
14. S.Goldwasser, S.Micali and C.Rackoff: *The Knowledge Complexity of Interactive Proof Systems*, SIAM J.Computing, Vol.18, pp.186-208, 1989.
15. J. van de Graaf and R.Peralta: *A simple and Secure Way to Show the Validity of your Public Key*, Proc. of Crypto 87, Lecture Notes in Computer Science, Springer Verlag.
16. Håstad and Shamir: *The Cryptographic Security of Truncated Linearly Related Variables*, Proc. of the ACM Symposion on the Theory of Computing, 1983, pp.356-362.
17. Impagliazzo and Yung: *Direct Minimum Knowledge Computations*, Proc. of Crypto 87, Lecture Notes in Computer Science, Springer Verlag.
18. Luby, Micali and Rackoff: *How to Simultaneously Exchange a Secret Bit by Flipping a Symmetrically-Biased Coin*, Proc. of the IEEE conference of the Foundations Of Computer Science 1983.
19. Rabin: *How to Exchange Secrets by Oblivious Transfer*, Tech. Memo, TR-81, Aiken Comp. Lab., Harward University, 1981.
20. Tedrick: *Fair Exchange of Secrets*, Proc. of Crypto 84, pp.434-438, Lecture Notes in Computer Science, Springer Verlag.
21. M.Tompa and H.Woll: *Random Self-Reducibility and Zero-Knowledge Proofs of Information Possession*, Proc. of FOCS 87.
22. Vazirani and Vazirani: *Trapdoor Pseudorandom Number Generators With Applications to Cryptographic Protocol Design*, Proc. of the IEEE conference on the Foundations Of Computer Science 1983, pp.23-30.
23. M.Waidner, B.Pfitzmann: *The Dining Cryptographers at the Disco: Unconditional Sender and Recipient Untraceability with Computational Secure Servicability*, Proc. of EuroCrypt 89, Lecture Notes in Computer Science, Springer Verlag.
24. Yao: *How to Generate and Exchange Secrets*, Proc. of the IEEE conference on the Foundations Of Computer Science 1986.