GRADUAL AND VERIFIABLE RELEASE OF A SECRET

(Extended Abstract)

Ernest F. Brickell

Bell Communications Research;
435 South Street; Morristown, NJ 07960; USA

David Chaum
Ivan B. Damgård
Jeroen van de Graaf

Centre for Mathematics and Computer Science
Kruislaan 413; 1098 SJ Amsterdam; The Netherlands.

Abstract:
Protocols are presented allowing someone with a secret discrete logarithm to release it, bit by bit, such that anyone can verify each bit's correctness as they receive it. This new notion of *release of secrets* generalizes and extends that of the already known *exchange of secrets* protocols. Consequently, the protocols presented allow exchange of secret discrete logs between any number of parties.

The basic protocol solves an even more general problem than that of releasing a discrete log. Given any instance of a discrete log problem in a group with public group operation, the party who knows the solution can make public some interval I and convince anyone that the solution belongs to I, while releasing no additional information, such as any hint as to where in I the solution is.

This can be used directly to release a discrete log, or to *transfer* it securely between different groups, i.e. prove that two instances are related such that knowledge of the solution to one implies knowledge of the solution to the other.

We show how this last application can be used to implement a more efficient release protocol by transferring the given discrete log instance to a group with special properties. In this scenario, each bit of the secret can be verified by a single modular squaring, and unlike the direct use of the basic protocol, no interactive proofs are needed after the basic setup has been done.

Finally, it is shown how the basic protocol can be used to release the factorization of a public composite number.

C. Pomerance (Ed.): Advances in Cryptology - CRYPTO '87, LNCS 293, pp. 156-166, 1988.

1. Related work

Two of the first contributions to the field of *secrets exchange* were made by Blum[BL81] and Rabin[Ra81]. They both showed how to exchange secrets using oblivious transfer. Later, Blum[Bl83] proposed his well known protocol for exchange of factorizations; see Hastad and Shamir [HaSh85] for a discussion of some problems with this protocol. It cleverly expoits the fact that the factorization of an integer is a secret of many bits. In the original version, one participant would sometimes know one secret bit more than the other. Tedrick's work [Te83], [Te84] is one approach to reducing the amount of unfairness introduced by this fact. A different approach to exchanging one bit was taken by Luby, Micali and Rackoff [LMR83] and Vazirani and Vazirani [VaVa83] Here, the information exchanged is probabilistic, i.e. in each round of the protocol additional certainty is gained about the value of a single secret bit. The work of Yao [Ya86] implies the existence of extremely general two party exchange-of-secret protocols, in connection with the problem of secure computations with private inputs.

While the practical proposals in the literature involve disclosure of secret factorizations (combined with square roots), the present protocol allows disclosure of discrete logarithms. This is of practical importance for protocols based on secret discrete logs, since means to *transfer* a secret from a discrete log form to a factorization form are not known.

Perhaps a more fundamental difference between the related literature and the present protocol, however, is that they only provide for *exchange* of secrets, while we allow the more general *release* of secrets (one possible exception is [VaVa83]. Still, the fact that the information released by our protocol is deterministic, remains a fundamental difference between the protocols). In earlier exchange of secrets protocols, it was either inherent in the protocol that an exchange had to take place [LMR83], or the work done by the party with the secret would increase rapidly with the number of parties who were to recieve it [Bl83]. In a release protocol the party with the secret is essentially just broadcasting information about it. Therefore, a release can be simultaneous to any number of parties, and can be readily used to implement an exchange of secrets between any number of parties.

The theoretical existence of of such release-of-secrets protocols can be proved using techniques from the work of Brassard and Crepeau [BrCr86], of Chaum [Ch86] and of Goldreich, Micali and Wigderson [GMW86]. But since their methods (as Yao's [Ya86]) involve individual processing of all gates in a boolean circuit or a reduction to 3COL, the resulting protocols would be impractical. We propose a more practical solution to the problem: in the most efficient version, to release one bit, one modular square root has to be computed, while the correctness of that bit can be checked by one squaring.

2. Showing membership of an interval.

Suppose one party — the prover (P) — knows a solution z to the equation $\alpha^z = \beta$ where α and β belong to some group with a public group operation. One obvious example of this is the multiplicative subgroup of the integers modulo p, denoted as Z_p^*, where p is a large prime. Suppose also that $z \in [a, a+B]$, where a and B are public. The protocol below can then be used by the prover to convince someone else — the verifier (V) — that $y \in [a-B, a+2B]$.

Although we talk about a single verifier for convenience, it should be noted that any set of participants could easily play the part of V without loosing the efficiency of the protocol. This is so because the role of V is passive: he flips coins (in public) and expects answers from P according to the coinflips. Thus, this protocol has the basic properties we require from a release of secrets protocol.

An important tool used in the protocol is a *bit commitment scheme.* When a party P commits to a bit, this means that he encrypts a 0 or 1 in such a way, that

- the other party cannot tell what the value of the bit is;
- P cannot later change his mind about the bit.
- P can convince anyone about the value of the bit. This is referred to as *opening* the commitment.

For the purpose of the protocol presented in this paper, it is not necessary to fix a particular choice of bit commitment scheme. The above properties will be sufficient to see why the protocol below is minimum knowledge. In [CDG87] bit commitment schemes are discussed in more detail; also a specific example is given that relies on the hardness of the discrete log problem. More precisely, with this scheme a commitment releases no information in the Shannon sense about the bit it contains, and the party committing to a bit cannot change it, unless he can solve a discrete log problem *before the protocol is over.* With this scheme, the protocol below will be *perfect zero knowledge* in the terminology of [GMW86] i.e. the secret of the prover is unconditionally protected, but the verifier will only be convinced modulo his belief that the prover has not been able to solve the hard problem that prevents him from cheating. Note that this flavour of commitment scheme does not fit into the original model of [GMR85], where the prover has infinite computing power. The term "perfect zero knowledge" therfore only referres to the fact that a simulation will produce the exact same probability distribution as the actual protocol. Other choices of commitment schemes (like one based only on the existence of a one-way permutation) will imply that the verifier will always be convinced with an exponentially small residue of doubt, and that the protocol will be "ordinary" zero knowledge, even with an infinitely powerful prover. In this case, however, the secret of the prover will only be conditionally protected, and will be revealed if the cryptographic assumption is broken at any point after completion of the protocol. For a more detailed discussion of these problems, consult [BCC87].

Having chosen a bit commitment scheme, the prover can commit to a string of bits by

simply computing commitments bit by bit. For the bitstring s, the resulting string of commitments is denoted $BC(s)$.

PROTOCOL RELEASE INTERVAL

1. P picks $t_1 \in [0,B]$ and computes $t_2 = t_1 - B$.
   P computes a bit commitment of α^{t_1} and of α^{t_2}, and sends the unordered pair $(BC(\alpha^{t_1}), BC(\alpha^{t_2}))$ to V.

2. V flips a coin and requests either
   (a) to receive t_1 and t_2 and opening by P of both $BC(\alpha^{t_1})$ and $BC(\alpha^{t_2})$, or
   (b) to receive $(z + t_i)$ for $i = 1$ or 2, and opening by P of $BC(\alpha^{t_i})$

3. On request (a) P sends t_1 and t_2, and opens both commitments.
   On request (b) P sends either $(z + t_1)$ or $(z + t_2)$, whichever is in the interval $[a, a + B]$, and opens the corresponding bit commitment.

4. On request (a), V checks that the two commitments did indeed contain α^{t_1} and α^{t_2}, that $t_1 \in [0,B]$, and that $t_1 - t_2 = B$.
   On request (b), V checks that α^{z+t_i} equals $\beta\alpha^{t_i}$ (he knows α^{t_i} as a result of the opening of one of the commitments).

Steps 1-4 are repeated n times.

Lemma 1: If the prover cannot change the contents of his commitments after step 1, and is able to satisfy both request (a) and request (b), then $z \in [a - B, a + 2B]$.

Proof: The result follows immediately from $t_1 \in [0,B]$, $t_2 \in [-B,0]$, and $z + t_i \in [a, a + B]$ for either $i = 1$ or 2. □

By this lemma, if the protocol is repeated n times, then the verifier will be convinced with confidence $1 - 2^{-n}$, assuming that the prover could not change the content of his bit commitments. As mentioned before, changing the content of a commitment is either impossible or occurs with only negligible probability, depending on the choice of bit commitment scheme.

Note that there is a discrepancy between the requirement for z, namely $z \in [a, a + B]$, and what is actually proven, namely $z \in [a - B, a + 2B]$. Below we will show that only if $z \in [a, a + B]$, no knowledge is released by the above protocol.

There are several related definitions for minimal knowledge, which are quite technical. In the full paper, we will give all of the formal definitions and the proof that our protocol is minimum knowledge, but for this extended abstract, we will only give intuitive definitions and proofs.

The concept of zero knowledge proofs was introduced by [GMR85]. Rougly speaking, they defined *knowledge* to be something that cannot be computed in random polynomial time. Essentially, they call a proof that a string x is in a language L, a *zero knowledge proof* if a probabilistic polynomial time turing machine will be convinced (with high

probability) that $x \in L$, but will learn nothing other than this one bit of knowledge.

To describe what we know about the discrete logarithm release protocol, we need to change the [GMR85] definition a little: First we will, as in [GHY87] and [FFS87], assume that both the prover and the verifier are restricted to polynomial time computations. This corresponds better to a practical situation, and allows use of both types of bit commitment schemes mentioned above. Secondly, we will say that a protocol is a (L_1,L_2) minimum knowledge protocol if the verifier will be convinced (with high probability) that $x \in L_1$ but the verifier will receive no knowledge other than the fact that $x \in L_2$. More precisely, the definition is satisfied if it is possible, assuming only that $x \in L_2$, to simulate a protocol conversation such that no probabilistic polynomial time algorithm can tell the difference between a simulated and a genuine conversation. This definition allows for a discrepancy between what the verifier is told and what he will be convinced of. It reduces to the [GMR] definition of zero knowledge when $L_1 = L_2$, apart from the assumption on the computational power of the prover.

We are now ready to give our informal proof that the discrete logarithm release protocol is an (L_1,L_2) minimum knowledge protocol, where $L_1 = \{\alpha^x \mid x \in [a-B,a+2B]\}$ and $L_2 = \{\alpha^x \mid x \in [a,a+B]\}$.

Lemma 2: If $z \in [a,a+B]$, then the distribution of the value of $(z + t_i)$ sent in step 3(b) of RELEASE INTERVAL is independent of the value of z.

Proof: It is trivial to check, that if some number $d \in [a,a+B]$ is sent as $(z + t_i)$, then exactly one value in $[0,B]$ must have been chosen as t_1. So if t_1 is chosen uniformly in $[0,B]$, then $(z + t_i)$ must be uniform in $[a,a+B]$ □.

Lemma 3: There exists a simulator for RELEASE INTERVAL.

Proof: Using the following observation it is easy to see that the protocol can be simulated. Since the protocol proceeds in rounds, the simulator (P') can rewind (i.e. stop and go back to a previous state) the protocol if the simulation gets stuck at any point. Thus, we can essentially assume that the simulator knows in advance which choice the verifier will make in step 2. It is therefore trivial to compute a message for step 1 that will satisfy the verifier if he is going to make choice (a). For choice (b) the simulator chooses $d \in [a,a+B]$ at random, to serve as $z+t_i$. It then puts $\gamma = \alpha^d\beta^{-1}$. Now P' can commit to γ, and to something totally random that has the same number of bits as γ. These commitments are sent in step 1; in step 3, d is sent in place of $z + t_i$, and the bit commitment containing γ is opened.

If unconditionally secure bit commitments are used, it is now clear from Lemma 2 that the simulated conversation will have exactly the same distribution as the actual protocol conversation.

If the other type of commitments mentioned above is used, i.e. a scheme based on probabilistic encryption, the distributions will be different, but the difference cannot be recognised in polynomial time, if the encryption function is secure against polynomial time attacks. □

P is of course free to choose $z \in [a+B, a+2B]$, but then the protocol releases information. For instance, if z is almost equal to $a+2B$, P must always choose t_1 close to 0, and release $(z+t_2)$ which is close to $a+B$. We could, however, take away P's freedom to choose t_1 himself. The protocol could start by a sequence of coinflips with the property that both parties trust their randomness, but only P gets to see the outcome. This can easily be implemented using bit commitments. The protocol would now proceed as before, but in step 3(a), P would have to prove that he computed t_1 from the coinflips generated initially. This modification means that there is now a non-zero probability of catching P if z is not in $[a, a+B]$. This probability is larger, the further away z is from $[a, a+B]$, and will tend to 0 with increasing n for any fixed z not in $[a, a+B]$. In fact, this means that we can get exponentially large confidence about any interval that properly contains $[a, a+B]$, at the cost of a larger n. More details will be given in a full version of this paper.

2.1 Release Interval Without Bit Commitments.

There is a variation of RELEASE INTERVAL that does not require the use of bit committment. This variation makes use of a parameter δ which we will assume is equal to ϵB for some $\epsilon \in (0,1)$. The protocol proceeds as follows:

RELEASE INTERVAL II

The prover

1. Picks $t \in [B, B+\delta]$
2. picks $t_1 \in [0,t]$ and computes $t_2 = t_1 - t$.
3. permutes t_1 and t_2.
4. Sends the unordered pair α^{t_1} and α^{t_2} to the verifier.

The verifier

5. Requests either (a) to receive t_1 and t_2 or (b) to receive $z+t_i$ for $i = 1$ or 2.

The prover

6. On request (a) sends t_1 and t_2. On request (b) sends either $z+t_1$ or $z+t_2$, whichever is in the interval $[a, a+t]$.

The verifier checks

7. on request (a), that the prover did indeed send α^{t_1} and α^{t_2}, and that $|t_1 = t_2| \in [B, B+\delta]$.
8. on request (b), that α^{z+t_i} equals either $\beta\alpha^{t_1}$ or $\beta\alpha^{t_2}$, and that $z+t_i \in [a, a+B+\delta]$.

After RELEASE INTERVAL II has been repeated n times, the verifier will be convinced with

confidence $1-\frac{1}{2^n}$ than $z \in [a-B-\delta, a+2B+2\delta]$, but this interval can be shortened if the verifier trusts the randomness of t and t_1. RELEASE INTERVAL II is more efficient than RELEASE INTERVAL. Further, assuming that

— the verifier does not have a probabilistic polynomial time algorithm that solves discrete log given that the solution is in an interval of size B

— $z \in [a, a+B]$,

then RELEASE INTERVAL II is minimum knowledge (but clearly not perfect zero knowledge). In other words, if the verifier cannot find the secret directly from the public information, then he is in essentially the same situation after having executed the protocol. The proof of this is quite long and technical and has therefore been omitted in this extended abstract.

3. Two applications

It is easy to see, how the RELEASE INTERVAL protocol can be used to release information about the solution to a discrete log problem in a verifiable way, starting with the high order bits: Use protocol RELEASE INTERVAL many times with ever decreasing values of B. This convinces the verifier that the solution is contained in smaller and smaller intervals.

If it is easy to compute square roots in the group involved, we can also release the solution, starting with the low order bits. Consider for instance Z_p^*, with p prime. Fix some generator α of Z_p^*. For any square $\beta \in Z_p^*$, we let the principal square root of β be the root with discrete log base α smaller than $\frac{p-1}{2}$. It is easy to see that the discrete log of any number can be computed bit by bit, starting with the low order bits, given an oracle that decides which of two square roots is the principal one. At each point in this computation, someone who already knows the discrete log could serve as the oracle by using RELEASE INTERVAL to prove the validity of his answers. This clearly leads to a protocol that releases the discrete log starting with the low order bits.

4. An Efficiency Improvement.

In the previous two applications we had to go the RELEASE INTERVAL once for each bit released. This can be quite time consuming in general because each instance of RELEASE INTERVAL involves a cut-and-choose with many iterations. Using a technique to transfer a discrete log from one group to another, we can reduce this to 1 application of RELEASE INTERVAL. However, besides the protocol needed to transfer the logarithm, we the need other protocols to prove that the groups used have specific properties. Recall that the

secret z is determined by $\alpha^z = \beta$, where $\alpha, \beta \in Z_p^*$. Basically we transfer the secret from $\langle\alpha\rangle$ to $\langle\alpha_1\rangle$, where $\alpha_1 \in Z_N^*$ and N is a composite. α_1 is chosen, such that 2^l divides $order(\alpha_2)$, where $l = {}^2\log p$. Then the secret is released just by showing square roots in Z_N^*, which are easily verified but hard to compute without the factorization. First we will show how P can prove to V that N has the required properties. This is done in steps 1-3 below. Note that these steps are only necessary once, i.e. if many secrets are to be released, the same N and α_1 can be used for all of them.

Step 1:
Party P chooses a modulus $N = qr$, where q and r are large primes, constructed such that P knows the factorization of $q-1$ and $r-1$, and such that 2^l divides at least one of $q-1$ and $r-1$. This can easily be done , e.g. by a variation of Gordon's method [Go84].

Step 2:
Let QR denote the set of quadratic residues modulo N. Next P selects a random element $\alpha_1 \in Z_N^*$ with $J(\frac{\alpha_1}{N}) = 1$ and α not in QR. From the Chinese Remainder Theorem it can easily be seen that such an α_1 will always have the maximum possible 2-power dividing its order, so 2^l is certainly a divisor of the order of α_1.

Step 3:
P makes public N and α_1. Then

a) P proves that α_1 is a quadratic non-residue, e.g. by using the protocol in [GMR85].
b) P proves that 2^l divides $order(\alpha_1)$:
i) V choses a random odd number r, with $0 < r < R$, where R is fixed (the choice of R is considered later). V chooses $b \in \{0,1\}$.
ii) V sends to P:
$\alpha_1^{r^{2^k}}$ if $b = 1$;
$\alpha_1^{r^{2^{k-1}}}$ if $b = 0$.
iii) Since P has constructed N, he can compute the order of what he receives. Based on the outcome he determines b and sends it to V.
iv) V checks that the correct value of b was returned.

Steps i through iv are repeated n times.

It is easy to see that if P is not cheating, he can always give the correct answer in step iii) Consider the situation where P is trying to cheat, i.e. the maximum 2-power dividing $order(\alpha_1)$ is 2^c, where $c < l$. Let G be the maximal sub-group of $\langle\alpha_1\rangle$ of odd order. Clearly, $G = \langle\alpha_1^{2^k}\rangle = \langle\alpha_1^{2^{k-1}}\rangle$, which means that both $\alpha_1^{r^{2^k}}$ and $\alpha_1^{r^{2^{k-1}}}$ will look to P like randomly chosen elements of G. Since V does not know the order of G, he cannot make the distribution completely uniform over G, but he can make as close an approximation as he likes by choosing the R from step i) large enough. Assuming that P cannot distinguish the distribution of $\alpha_1^{r^{2^k}}$ and $\alpha_1^{r^{2^{k-1}}}$, he can do no better than guesing at random in step iii) whence he will be caught with probability $1-2^{-n}$.

From V's point of view the protocol is not zero knowledge as it stands. V could use it to get some — probably useless — information about other elements in Z_N^* than α_1. The

protocol can be modified such that it is zero-knowledge. This involves ensuring that V knows the discrete log with respect to α_1 of the numbers he send to P. This could be accomplished using for example the general discrete log protocol described in [CEGP86]. We also have other methods which will work well in this special case, but we omit the details here for simplicity.

Step 4:
P transfers the problem from Z_p^* to Z_N^*. Remember P's secret is the solution z of $\alpha^z = \beta_1$ in Z_p^*. Now P computes $\beta_1 = \alpha_1^{\tilde{l}}$ in Z_N^* and makes β_1 public. Using RELEASE INTERVAL in the cross-product of $\langle\alpha\rangle$ and $\langle\alpha_1\rangle$, P can prove that he knows a $z \in [1,p-1]$, which solves both $\alpha^z = \beta$ and $\alpha_1^{\tilde{l}} = \beta_1$.

Step 5:
To release a single bit (the least significant bit of z), P proceeds as follows:

if $z = 2z'$, he makes public a square root of β_1.

if $z = 2z'+1$, he makes public a square root of $\beta_1\alpha_1^{-1}$.

Replace z by z', and β_1 by the square root released.

This step works because of the following two easily verified facts:

1. The l least significant bits of z are uniquely determined by the equation $\alpha_1^{\tilde{l}} = \beta_1$.
2. If α_1 is a non square mod N and $\alpha_1^{\tilde{l}} = \beta_1$, then z is even if and only if β_1 is a square mod N.

By fact 1, the pair (α_1,β_1) determines exactly one secret of size l bits, and by fact 2, anyone can check on P by squaring the numbers released.

Regarding the security of this protocol, notice first that steps 2 and 3 can be executed in zero knowledge. Moreover, the release of α_1 cannot endanger the factorization of N, since anyone can select at random a number of Jacobi symbol 1 mod N, which with probability ½ will have the same properties as α_1. It is clear that breaking this scheme can be no harder than factoring numbers of the required special form: $N = qr$, where 2^l divides at least one of $q-1$ and $r-1$. It is not known, however, whether these numbers are easier to factor than randomly chosen RSA-moduli. But with respect to the factoring algorithms known at present, the factorization of N should be safe, if $q-1$ and $r-1$ contain large prime factors, in addition to the 2-powers. Finally, it is also conceivable that someone could try to attack the equations $\alpha^z = \beta$ and $\alpha_1^{\tilde{l}} = \beta_1$ directly, without factoring N. But this involves solving a discrete log problem, and we know of no argument indicating that discrete log should be easier in this special case than in the general situation.

6. Releasing a factorization.

This final application allows a prover to release the factorization of a public modulus N. It is wellknown that knowledge of a multiple of $\phi(N)$ suffices to factor N easily. Using this fact, the protocol proceeds as follows: The verifier chooses an integer T and a set of elements $a_1, \ldots, a_s$ in Z_N^*. The prover then computes $t = T \bmod \phi(N)$, and sends $b_i = a_i^t$ to the verifier, who can check these numbers by raising the a's to the T'th power. The prover then uses the RELEASE INTERVAL protocol to convince the verifier that he knows a simultaneous solution to all s discrete log instances which belongs to an interval NOT including T. When the prover uses RELEASE INTERVAL more times to release bits of t, the verifier can be convinced, that he will get at least a multiple of $LCM(\text{order}(a_1),..,\text{order}(a_s))$. The probability that this number equals the exponent of the group Z_N^* (i.e. the smallest integer e such that $g^e = 1$ for all elements in the group) is exponentially large in s, and it is not hard to see that $exponent(Z_N^*)$ will serve as well in factoring N as $\phi(N)$.

Acknowledgement.

The first author would like to thank Oded Goldreich for discussions related to this work.

References

[Bl81] Blum: "Three applications of the oblivious transfer", Dept. of EECS, Univ. of California, Berkely, 1981.

[Bl83] Blum: "How to exchange (secret) keys", *ACM Transactions on Computer Systems*, vol. 1, 1983, pp. 175-193.

[BCC87] Brassard, Chaum and Crépeau:"Minimum disclosure proofs of knowledge", to appear.

[BrCr86] G. Brassard, and C. Crépeau, "Zero-Knowledge Simulation of Boolean Circuits," *Presented at Crypto 86*, (August 1986).

[Ch86] D. Chaum, "Demonstrating that a Public Predicate can be Satisfied Without Revealing Any Information About How," *Presented at Crypto 86*, (August 1986).

[CDG87] Chaum, Damgård and van de Graaf: "Multiparty computations ensuring privacy of each party's input and correctness of the result", Proc. of Crypto 87.

[CEGP86] D. Chaum, J.-H. Evertse, J. van de Graaf, and R. Peralta, "Demonstrating

possession of a discrete logarithm without revealing it," *To appear in the Proceedings of Crypto 86*, (August 1986).

[CG87] Chaum, van de Graaf: "An improved protocol for demonstrating possession of a discrete log and some generalisations", Proc. of Eurocrypt 87.

[FFS87] Fiege, Fiat and Shamir: "Zero knowledge proof of identity", Proc. of STOC 87.

[GHY87] Galil, Haber and Yung: "Cryptographic Computation: Secure Fault-tolerant Protocols in the Public Key Model", Proc. of Crypto 87.

[GMR85] S. Goldwasser, S.Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proof Systems," *17th STOC* (1985).

[GMW86] O. Goldreich, S. Micali, and A. Wigderson, "How to Prove all NP-statements in Zero-Knowledge, and a Methodology of Cryptographic Protocol Design," *Presented at Crypto 86*, (August 1986).

[Go84] J. Gordon, "Strong primes are easy to find", Proceedings of Eurocrypt 84.

[HaSh85] "The cryptographic security of truncated linearly related variables," *Proc. of 17th STOC*, 1985, pp. 356-362.

[LMR83] Luby, Micali and Rackoff: "How to simultaneously exchange a secret bit by flipping a symmetrically biased coin", Proc. 24th FOCS, 1983, pp.11-21.

[Ra81] Rabin: "How to exchange secrets using oblivious transfer", Technical memo, TR-81, Aiken Computation Lab., Harward Univ., 1981.

[VaVa83] Vazirani and Vazirani: "Trapdoor pseudo random number generators with applications to protocol design", Proc. 24th. FOCS, 1983, pp.23-30.

[Te83] Tedrick: "How to exchange half a bit", Proc. of Crypto 83, pp.147-151.

[Te84] Tedrick: "Fair exchange of secrets", Proc. of Crypto 84, pp.434-438.