J. Cryptology (1991) 4: 151–158

Journal of Cryptology

Bit Commitment Using Pseudorandomness[1]

Moni Naor
IBM Almaden Research Center, 650 Harry Road,
San Jose, CA 95120, U.S.A.

Abstract. We show how a pseudorandom generator can provide a bit-commitment protocol. We also analyze the number of bits communicated when parties commit to many bits simultaneously, and show that the assumption of the existence of pseudorandom generators suffices to assure amortized $O(1)$ bits of communication per bit commitment.

Key words. Cryptographic protocols, Pseudorandomness, Zero-knowledge proof systems.

1. Introduction

A bit-commitment protocol is a basic component of many cryptographic protocols. One party, Alice, commits to the other party, Bob, a bit b, in such a way that Bob has no idea what b is. At a later stage Alice can reveal the bit b and Bob can verify that this is indeed the value to which Alice committed. A good way to think about it is as if Alice writes the bit and puts it in a locked box to which only she has the key. She gives the box to Bob (the commit stage) and when the time is ripe she opens it. Bob knows that the contents were not tampered with, since the box was in his possession.

Bit commitment has been used for zero-knowledge protocols [GMW1], [BCC], identification schemes [FS], and multiparty protocols [GMW2], [CDG], and it can implement Blum's coin flipping over the phone [B].

A current research program in cryptography is to base security of protocols on as general assumptions as possible. Past successes of the program had been in establishing various primitives on the existence of one-way functions or permutations or on the existence of trapdoor functions. The most general (computational complexity) assumption under which bit commitment was known to be possible is that one-way permutations exist [GMW1]. In this paper we show that given any pseudorandom generator, a bit-commitment protocol can be constructed. This is a weaker condition, since Yao [Y] has shown that pseudorandom generators can be

[1] Date received: October 31, 1989. Date revised: May 9, 1990. Part of this work was done while the author was at the University of California at Berkeley. This research was supported by NSF Grant CCR 88-13632.

based on one-way permutations. A pseudorandom generator is a function that maps a string (the seed) to a longer one, such that if the seed is chosen at random, then the output is indistinguishable from a truly random distribution for all polynomial-time machines. Impagliazzo *et al.* [ILL] have shown that given any one-way function (not necessary a permutation), a pseudorandom generator can be constructed (under nonuniform assumptions), and Hastad [H] has shown the same under uniform assumptions. On the other hand, Impagliazzo and Luby [IL] have argued that the existence of one-way functions is a prerequisite for any protocol that must rely on computational complexity. Thus we can conclude that if any computational-complexity-based cryptography is possible, then bit-commitment protocols exist, and so do the protocols that rely on bit commitment, such as zero-knowledge proofs and identification schemes.

What is the communication complexity of a bit-commitment protocol (i.e., how many bits must be transferred during the execution of the protocol)? It cannot be the case that only a fixed number of bits will be exchanged during the execution of the protocol. If this were the case, then after the commit stage Bob could guess with nonnegligible probability what Alice would send in the revealing stage, and can verify that the guess is consistent with what she sent so far and deduce the value of the bit. However, in many applications Alice wants to commit to a collection of bits $b_1, b_2, \ldots, b_m$ and they are to be revealed at the same time. These applications include coin flipping over the phone and zero-knowledge protocols as in [IY]. Furthermore, Kilian *et al.* [KMO] have shown that many of the known zero-knowledge protocols can be converted to ones that have this property. Therefore it is desirable to amortize the communication complexity of bit commitment. We show that if m is large enough, at least linear in the security parameter n, then Alice can commit to $b_1, b_2, \ldots, b_m$ while exchanging only $O(1)$ bits per bit commitment. The total computational complexity of the protocol is the same as the complexity of the protocol for committing to one bit.

In the next section we give formal definitions of the problem and the assumptions. In Section 3 we show how the commit can be implemented using a pseudorandom generator. Section 4 shows how to get the amortized communication complexity down to $O(1)$ per bit.

2. Definitions

A *bit-commitment* protocol consists of two stages:

- The *commit* stage: Alice has a bit b which she wishes to commit to Bob. She and Bob exchange messages. At the end of this stage, Bob has some information that represents b.
- The *reveal* stage: At the end of this stage, Bob knows b.

The protocol must obey the following: For all probabilistic polynomial-time Bobs, for all polynomials p and for large enough security parameter n:

1. After the commit stage Bob cannot guess b with probability greater than $\frac{1}{2} + 1/p(n)$.

2. Alice can reveal only one possible value. If she tries to reveal a different value she is caught with probability at least $1 - 1/p(n)$.

In defining the properties a bit-commitment protocol must obey, we have assumed a scenario where Bob cannot guess b with probability greater than 1/2 prior to the execution of the commit protocol. In the more general case, Bob has some auxiliary input that might allow him to guess b with probability $q > \frac{1}{2}$. In this case, the result of the commit stage is that Bob gains less than $1/p(n)$ advantage in guessing b. All the results of this paper hold for the general case.

A *commitment to many bits* protocol consists of two stages:

- The *commit* stage: Alice has a sequence of bits $D = b_1, b_2, \ldots, b_m$ which she wishes to commit to Bob. She and Bob exchange messages. At the end of this stage, Bob has some information that represents D.
- The *reveal* stage: At the end of this stage, Bob knows D.

The protocol must obey the following: For all probabilistic polynomial-time Bobs, for all polynomials p and for large enough security parameter n:

1. For any two sequences $D = b_1, b_2, \ldots, b_m$ and $D' = b'_1, b'_2, \ldots, b'_m$ selected by Bob, following the commit stage Bob cannot guess whether D or D' was committed with probability greater than $\frac{1}{2} + 1/p(n)$.
2. Alice can reveal only one possible sequence of bits. If she tries to reveal a different sequence of bits, then she is caught with probability at least $1 - 1/p(n)$.

Pseudorandom Generators

Let $m(n)$ be some function such that $m(n) > n$. G: $\{0, 1\}^n \mapsto \{0, 1\}^{m(n)}$ is a *pseudorandom generator* if, for all polynomials p and all probabilistic polynomial-time machines A that attempt to distinguish between outputs of the generator and truly random sequences, except for finitely many n's,

$$|\Pr[A(y) = 1] - \Pr[A(G(s)) = 1]| < \frac{1}{p(n)},$$

where the probabilities are taken over $y \in \{0, 1\}^{m(n)}$ and $s \in \{0, 1\}^n$ chosen uniformly at random.

Remark. We could have defined pseudorandom generators relative to polynomial-sized circuits. The results in this paper would change appropriately, i.e., the bit-commitment protocol would be secure against polynomial-sized circuits.

It is known that if pseudorandom generators exist for any $m(n) > n$, then they exists for all m polynomial in n [GGM]. We can treat the pseudorandom generator as outputting a sequence of unspecified length, of which we can examine only a fixed prefix (whose length is polynomial in n, the seed length).

In the rest of the paper we assume some psuedorandom generator G. Let n be a security parameter which is assumed to have been chosen so that no feasible machine can break the pseudorandom generator for seeds of length n. We use $G_l(s)$ to denote the first l bits of the pseudorandom sequence on seed $s \in \{0, 1\}^n$. $B_i(s)$ is used to denote the ith bit of the pseudorandom sequence on seed s.

3. The Bit Commitment

Pseudorandom sequences have the unpredictability of the next bit property: Given the first m bits of a pseudorandom sequence, any polynomial-time algorithm that tries to predict the next bit in the sequence has success probability smaller than $\frac{1}{2} + 1/p(n)$ for any polynomial $p(n)$. (In fact, Blum and Micali [BM] used this property to define pseudorandomness and Yao [Y] has shown that the two definitions are equivalent.) It is natural to try to apply this property to achieve bit commitment.

As a first attempt, consider the following protocol:

- Commit stage—Alice selects seed $s \in \{0, 1\}^n$ and sends $G_m(s)$ and $B_{m+1}(s) \oplus b$. (b is the bit Alice is committed to.)
- Reveal stage—Alice sends s, Bob verifies that $G_m(s)$ is what Alice send him before and computes $b = B_{m+1}(s) \oplus (B_{m+1}(s) \oplus b)$.

This protocol has the property that Bob cannot guess the bit that Alice commits to before the revealing stage, except with probability smaller than $\frac{1}{2} + 1/\text{poly}(n)$, because he does not have the power to predict the pseudorandom sequence. On the other hand, Alice might be able to cheat: if she finds two seeds s_1 and s_2 such that $G_m(s_1) = G_m(s_2)$, but $B_{m+1}(s_1) \neq B_{m+1}(s_2)$, then she can reveal any bit she wishes (by sending s_1 or s_2). There is nothing in the definition of pseudorandom generators that forbids the existence of such pairs. Furthermore, given any pseudorandom generator G, another pseudorandom generator G' that has such pairs can be constructed.

There is no way to force Alice to stick to one seed, since there may be two seeds that yield the same sequence. However, what the following protocol does is to force Alice to stick to the same *pseudorandom sequence*, or she will be caught with high probability.

Bit-Commitment Protocol.

- Commit stage—
  1. Bob selects a random vector $\mathbf{R} = (r_1, r_2, r_{3n})$ where $r_i \in \{0, 1\}$ for $1 \leq i \leq 3n$ and sends it to Alice.
  2. Alice selects a seed $s \in \{0, 1\}^n$ and sends to Bob the vector $\mathbf{D} = (d_1, d_2, \ldots, d_{3n})$ where
$$d_i = \begin{cases} B_i(s) & \text{if } r_i = 0, \\ B_i(s) \oplus b & \text{if } r_i = 1. \end{cases}$$
- Reveal stage—Alice sends s and Bob verifies that, for all $1 \leq i \leq 3n$, if $r_i = 0$, then $d_i = B_i(s)$, and if $r_i = 1$, then $d_i = B_i(s) \oplus b$.

This protocal maintains the property that Bob learns nothing about the bit b. Otherwise Bob would have the power to distinguish between outputs of the pseudorandom generator and truly random strings: If Alice had chosen a truly random sequence instead of a pseudorandom sequence, then Bob would not have learned anything about b, since all vectors $\mathbf{D}$ are equally likely, no matter what b is. (This

is still true even in the general case where Bob has some auxiliary input that allows him to guess b with probability $q > 1/2$.) If there exists a probabilistic polynomial-time Bob (call him Bob') that can learn something about b when Alice uses a pseudorandom sequence, then Bob' can be used to construct a distinguisher between outputs of G and truly random sequences. Given a sequence x, run the commit stage of the protocol with Alice and Bob', where Alice commits to a random b and instead of creating a pseudorandom sequence uses x. Let Bob' guess b. If he guesses correctly decide that x is pseudorandom, otherwise decide that x is truly random. The difference in the probability of deciding that the sequence is pseudorandom between a random sequence and a pseudorandom sequence is equal to the advantage Bob' has of guessing b in case x is a pseudorandom sequence.

How can Alice cheat? Her only chance to cheat is if there exist two seeds s_1 and s_2 such that $G_{3n}(s_1)$ and $G_{3n}(s_2)$ agree in all positions i where $r_i = 0$, and totally disagree in all positions i where $r_i = 1$. We say that such a pair fools **R**.

Claim 3.1. *The probability that there exists a pair of seeds s_1 and s_2 that fools* **R** *is at most 2^{-n}, where the probability is taken over the choices of* **R**.

Proof. If a pair s_1, s_2 fools **R**, then we know that $r_i = B_i(s_1) \oplus B_i(s_2)$. Therefore, a pair s_1 and s_2 fools exactly one **R**. There are 2^{2n} pairs of seeds and 2^{3n} vectors **R**. Hence the probability that there exists a pair that can fool the **R** that Bob chose is at most $2^{2n}/2^{3n} = 2^{-n}$. □

We can summarize by

Theorem 3.1. *If G is a pseudorandom generator, then the bit-commitment protocol presented obeys the following: For all polynomials p and for large enough security parameter n:*

1. *Following the commit stage, no probabilistic polynomial-time Bob can guess b with probability greater than $\frac{1}{2} + 1/p(n)$.*
2. *Alice can reveal only one possible bit, except with probability less than 2^{-n}.*

4. Efficient Commitment to Many Bits

The protocol given in the previous section has a communication cost of $O(n)$ bits. If Alice wants to commit to many bits $b_1, b_2, \ldots, b_m$ which she will reveal simultaneously, then she can do better. The idea is to use many bits to force Alice to stick to one pseudorandom sequence and use that sequence to commit to many bits.

Suppose we implement a protocol similar to the one in the previous section, but Bob, instead of requesting to see part of the pseudorandom sequence Xored with b, asks to see its bit-wise Xor with $b_1, b_2, \ldots, b_m$. (We assume here that $m = \frac{3}{2}n$ and that **R** contains exactly $\frac{3}{2}n$ ones.) Alice might be able to alter one of the b_i's, since it is enough that there exists a pair of seeds that agree on all the bits but one.

We prevent this form happening by using error-correcting codes with a large

distance between code words. Let $C \subset \{0,1\}^q$ be a code of 2^m words such that the hamming distance between any $c_1, c_2 \in C$ is at least $\varepsilon \cdot q$. We also require that there be an efficiently computable function E: $\{0,1\}^m \mapsto \{0,1\}^q$ for mapping words in $\{0,1\}^m$ to C.

What is required of the code? As we shall see, $q \cdot \log(2/(2-\varepsilon))$ must be at least $3n$, and we want q/m to be a fixed constant. Such codes exist, and specifically the Justesen code is a constructive example [J]. For the amortization to work it is sufficient that m be linear in n.

For a vector $\mathbf{R} = (r_1, r_2, \ldots, r_k)$ with $r_i \in \{0,1\}$ and with exactly q indices i such that $r_i = 1$, let $G_{\mathbf{R}}(s)$ denote the vector $\mathbf{A} = (a_1, a_2, \ldots, a_q)$ where $a_i = B_{j(i)}(s)$ and $j(i)$ is the index of the ith 1 in $\mathbf{R}$. If $e_1, e_2 \in \{0,1\}^q$, then $e_1 \oplus e_2$ denotes the bitwise Xor of e_1 and e_2.

Commitment to Many Bits Protocol. Alice commits to $b_1, b_2, \ldots, b_m$.

- Commit stage—
  1. Bob selects a ramdom vector $\mathbf{R} = (r_1, r_2, \ldots, r_{2q})$ where $r_i \in \{0,1\}$ for $1 \leq i \leq 2q$ and exactly q of the r_i's are 1 and sends it to Alice
  2. Alice computes $c = E(b_1, b_2, \ldots, b_m)$. Alice selects a seed $s \in \{0,1\}^n$ and sends to Bob $e = c \oplus G_{\mathbf{R}}(s)$ (the bitwise Xor of $G_{\mathbf{R}}(s)$ and c), and for each $1 \leq i \leq 2q$ such that $r_i = 0$ she sends $B_i(s)$.
- Reveal stage–Alice sends s and $b_1, b_2, \ldots, b_m$. Bob verifies that for all $1 \leq i \leq 2q$ such that $r_i = 0$ Alice had sent the correct $B_i(s)$, computes $c = E(b_1, b_2, \ldots, b_m)$ and $G_{\mathbf{R}}(s)$, and verifies that $e = c \oplus G_{\mathbf{R}}(s)$

As in the previous section, Bob can learn nothing about any of the b_i's:

Claim 4.1. *For any two different sequences $D = b_1, b_2, \ldots, b_m$ and $D' = b'_1, b'_2, \ldots, b'_m$ that Bob selects, for any polynomial p, following the commit stage Bob cannot decide with probability greater than $\frac{1}{2} + 1/p(n)$ to which sequence Alice has committed.*

Proof. If instead of a pseudorandom sequence Alice uses a truly random sequence, then the distribution that Bob sees is identical no matter what sequence of bits is being committed to. Thus if he can find two sequences of bits D and D' for which he can distinguish between D and D', then he has a distinguisher to the pseudo-random generator. □

How can Alice cheat? She can cheat if there exists a pair of seeds s_1 and s_2 that agree on all the indices that $\mathbf{R}$ has a 0, and there exist two different sequences $b_1, b_2, \ldots, b_m$ and $b'_1, b'_2, \ldots, b'_m$ such that $G_{\mathbf{R}}(s_1) \oplus E(b_1, b_2, \ldots, b_m) = G_{\mathbf{R}}(s_2) \oplus E(b'_1, b'_2, \ldots, b'_m)$. We say that s_1 and s_2 fool $\mathbf{R}$ in this case.

Claim 4.2. *For any pair of seeds s_1 and s_2, the probability that it fools $\mathbf{R}$ is at most $(1 - \varepsilon/2)^q$, where the probability is taken over the choices of $\mathbf{R}$.*

Proof. If s_1 *and* s_2 can fool some $\mathbf{R}$, then the hamming distance between $G_{2q}(s_1)$ and $G_{2q}(s_2)$ must be at least εq, since $G_{\mathbf{R}}(s_1) \oplus e = c_1$ and $G_{\mathbf{R}}(s_2) \oplus e = c_2$ for two

different code words c_1 and c_2 whose distance is at least εq. Therefore, the probability that the indices i for which $r_i = 0$ will hit only the indices where $G_{2q}(s_1)$ and $G_{2q}(s_2)$ agree is at most $((2q - \varepsilon q)/2q)^q = (1 - \varepsilon/2)^q$. □

If $(1 - \varepsilon/2)^q < 2^{-3n}$, i.e., $q \cdot \log(2/(2 - \varepsilon)) > 3n$, then for at most 2^{-n} of the vectors $\mathbf{R} \in \{0, 1\}^{2q}$ there is a pair of seeds s_1 and s_2 that fools $\mathbf{R}$. Therefore, the probability that Alice can alter any bit without being caught is at most 2^{-n}.

The number of bits exchanged in he protocol is $O(q)$, and when amortized over m bits it is $O(q/m)$ which is $O(1)$, since C is a good code. The dominant factor in the computational complexity of the protocol is that of G. Alice has to produce a psuedorandom sequence of length $2q$ which is $O(n)$. This is similar to the complexity of the commitment to a single bit.

We can summarize by

Theorem 4.1. *If G is a pseudorandom generator, then the many-bit-commitment protocol presented obeys the following: for all probabilistic polynomial time Bobs, for all polynomials p and for large enough security parameter n:*

1. *For any two different sequences $D = b_1, b_2, \ldots, b_m$ and $D' = b'_1, b', \ldots, b'_m$ that Bob selects, for any polynomial p, following the commit stage Bob cannot decide with probability greater than $\frac{1}{2} + 1/p(n)$ to which sequence Alice has committed.*
2. *Alice can reveal only one possible sequence of bits, except with probability less than 2^{-n}.*
3. *For $m > n$, the communication cost is $O(m)$.*

Joe Kilian (private communication) has suggested a different method for amortizing the communication complexity: commit to a seed s by committing to each of its bits separately and then commit to $b_1, b_2, \ldots, b_m$ by providing its Xor with the pseudorandom sequence generated by s. However, in this method the amortization starts only when m is at least n^2.

5. Conclusions

We have shown how to construct bit-commitment protocols from pseudorandom generators and have shown how bit commitment to many bits can be implemented very efficiently. Thus, various zero-knowledge protocols can be implemented with low complexity under the sole assumption that one-way functions exist.

In both protocols we have presented, Bob selects a random $\mathbf{R}$, and we have argued that almost all the $\mathbf{R}$'s are good. Therefore if there is a trusted party at some point in time (say the protocol designer), it can choose $\mathbf{R}$ and the same $\mathbf{R}$ will be used in all executions of the protocol.

References

[B] M. Blum, Coin flipping by telephone, *Proc. 24th IEEE Compcon*, 1982, pp. 133–137.

[BM] M. Blum and S. Micali, How to generate cryptographically strong sequences of pseudo-random bits, *SIAM Journal on Computing*, Vol. 13 (1984), pp. 850–864.

[BCC] G. Brassard, D. Chaum, and C. Crépeau, Minimum disclosure proofs of knowledge, *Journal of Computer and System Sciences*, Vol. 37 (1988). pp. 156–189.

[CDG] D. Chaum, I. Damgård, and J. van de Graaf, Multiparty computations ensuring secrecy of each party's input and correctness of the output, *Proc. Crypto '87*, p. 462.

[FS] A. Fiat and A. Shamir, How to prove yourself, *Proc. Crypto '86*, pp. 641–654.

[GGM] O. Goldreich, S. Goldwasser, and M. Micali, How to construct random functions, *Journal of the ACM*, Vol. 33 (1986), pp. 792–807.

[GMW1] O. Goldreich, M. Micali, and A. Wigderson, Proofs that yield nothing but their validity and a methodology of cryptographic protocol design, *Proc. 27th IEEE Symposium on Foundations of Computer Science*, 1986, pp. 174–187.

[GMW2] O. Goldreich, M. Micali, and A. Wigderson, How to play any mental game, *Proc. 19th ACM Symposium on Theory of Computing*, 1987, pp. 218–229.

[H] J. Hastad, Pseudorandom generators under uniform assumptions, *Proc. 22nd ACM Symposium on Theory of Computing*, 1990, pp. 395–404.

[ILL] I. Impagliazzo, L. Levin, and M. Luby, Pseudorandom generation from one-way functions, *Proc. 21st ACM Symposium on Theory of Computing*, 1989, pp. 12–24.

[IL] I. Impagliazzo and M. Luby, One-way functions are essential to computational based cryptography, *Proc. 30th IEEE Symposium on Foundations of Computer Science*, 1989, pp. 230–235.

[IY] R. Impagliazzo and M. Yung, Direct zero-knowledge protocols, *Proc. Crypto '87*, pp. 40–51.

[J] J. Justesen, A class of constructive asymptotically good algebraic codes, *IEEE Transactions on Information Theory*, Vol. 18 (1972), pp. 652–656.

[KMO] J. Kilian, S. Micali, and R. Ostrovsky, Minimum resource zero-knowledge proofs, *Proc. 30th IEEE Symposium on Foundations of Computer Science*, 1989, pp. 474–479.

[Y] A. C. Yao, Theory and applications of trapdoor functions, *Proc. 23rd Symposium on Foundations of Computer Science*, 1982, pp. 80–91.