ISSN 0105-8517

A Discrete Logarithm Blob for Noninteractive XOR Gates

Joan Boyar
Ivan Damgård

DAIMI PB – 327
August 1990

ISSN 0105-8517

PB – 327 **Boyar & Damgård: A Discrete Logarithm Blob ...**

A Discrete Logarithm Blob for Noninteractive XOR Gates

Joan Boyar[*]
Aarhus University

Ivan Damgård
Aarhus University

August 1, 1990

1 Introduction

Bit commitments, also known as blobs, have been a very important part of all zero-knowledge interactive proofs and interactive arguments for problems which are NP-complete. Thus there have been many proposals for bit commitment schemes. Those which have been based on complexity-theoretic assumptions have, with only a few exceptions, been based on either the intractability of factoring or the intractability of the discrete logarithm. Since it is unknown which assumption is better, both types of implementations are interesting.

Different bit commitment shemes are appropriate for different applications. For example, some schemes lead to interactive proofs which are unconditionally secure for the verifier, i.e. the prover will be unable to cheat even if the cryptographic assumptions fail. With other schemes, one gets interactive arguments which are unconditionally secure for the prover, i.e. the verifier will have no information whatsoever even if the cryptographic assumptions fail.

Another example of the difference in applicability of different schemes is that, although the discrete logarithm implementation in [1] and [6] is

[*]Supported in part by the ESPRIT II Basic Research Actions Program of the EC under contract No. 3075 (Project ALCOM).

the only scheme that leads to perfect zero-knowledge interactive arguments, its usage is limited by the fact that it is possible for the prover to create a pair of blobs for which it can "prove" that they are encryptions of the same bit and encryptions of different bits, but for which it can open neither of them. Fortunately, in many applications either there is no equality or inequality checking, or, for any pair of blobs that the verifier may choose to have compared, the verifier also has the option of asking that one of these blobs be opened [4].

In addition, with certain bit commitment schemes based on factoring, for a set of blobs which are the commitments to some bits, it is possible to create a blob which represents the exclusive or of those bits simply by multiplying the blobs [5] [2]. There does not appear to be any way to do such noninteractive processing of exclusive or gates using the discrete logarithm blobs of [1] and [6].

In this paper, we present a bit commitment scheme based on the discrete logarithm, which allows noninteractive processing of XOR and NOT gates. This implementation leads to interactive arguments which are statistical zero-knowledge.

2 The blob

Initially, the verifier chooses a random factored number n such that $p = n + 1$ is a prime, a generator g, and an element a of the group Z_p^*. The verifier sends the prover p, g, a, and the factorization of n,[1] so the prover can check that g is a generator. The blob for a bit b is

$$a^s g^r mod\ p \tag{1}$$

where r is uniform between 0 and $p-1$, and either s is uniform on the even integers between 0 and S, or on the odd integers between 0 and S. S is a public constant chosen such that it is exponentially large, but still small compared to p (e.g. $p^{\cdot 1}$). And finally, s is even if and only if $b = 0$. To open a blob, the prover shows s and r, where of course s must be smaller than S.

Some restriction on the size of s is necessary since otherwise the prover could easily cheat, changing zero blobs to one blobs and vice versa at will.

[1]This factorization could be replaced by a zero-knowledge interactive proof that a is in the subgroup generated by g.

Suppose that 2^t is the largest power of 2 dividing $p-1$. It is easy to find u such that $a^{(p-1)/2^t} g^{(p-1)/2^u} = 1 \; mod \; p$. If a blob is created as $a^s g^r mod \; p$, if there was no restriction on the size of s, it could also be expressed as $a^{(s+(p-1)/2^t) mod \; p-1} g^{(r+(p-1)/2^u) mod \; p-1}$, changing the parity of the exponent on a. This is impossible if that exponent must always be "small".

Notice that the "blinding factor" g^r prevents the verifier from determining the parity of s simply from the quadratic residuosity of the blob. This blinding factor makes the blob a random element of the subgroup generated by g (which is the entire group if g is a generator), and thus a blob carries no information about the bit it encodes.

If the prover can open a blob in two ways, then this amounts to knowing an equation of the form $a^x = g^y$, where x has absolute value smaller than S. The "worst" thing that can happen is that x divides $p-1$, but since x is relatively small, the prover must still obtain non-trivial information about the discrete log of a in order to cheat. If the exponent x is restricted to having an absolute value less than the smallest odd prime factor of $p-1$, then this is equivalent to knowing the discrete log of a.

3 XORing and Negating Bits

It is clear that the blob on the output wire from a NOT gate can be computed from the blob on the input, simply by multiplying by a.

At first glance, it also appears that all we have to do to compute output blobs for XOR gates with any number of inputs is to multiply the input blobs together, since this will add the s-values.

This works to a certain extent: to "XOR" n blobs, we multiply them, and interpret the product as a blob in the same way as before. If the product is to be opened, we accept the opening, if the s-value is less than nS. The basic result for this procedure is:

Theorem 1

Let $B_1, ..., B_n$ be blobs created by the prover according to the above definition. Then $log(p)$ can be chosen polynomial in n such that the following holds: The product modulo p, $B_1 \cdots B_n$, is a blob that the prover can open in at most one way. Suppose the prover knows how to open $B_1, ..., B_n$, yielding bits $b_1, ..., b_n$, resp. Then he can only open the product as representing $b_1 \oplus \cdots \oplus b_n$, and the opening will release only an exponentially small (in $log(p)$) amount of additional information about $b_1, ..., b_n$.

Proof

Suppose the prover could open the product in two different ways. Then as in the previous section, this means he knows an equation of the form $a^x = g^y$, where x has absolute value less than nS. Again, this means that the prover has obtained nontrivial information about the discrete log of a, since n is "small" in the sense of section 2.

For the second statement, let $s_1, ..., s_n$ be the s-values claimed by the prover for $B_1, ..., B_n$. Then it is clear that the prover can open the product using the sum of the s_i's to yield the bit $b_1 \oplus ... \oplus b_n$, and by the first statement, this is the only way he can open it. Moreover, given $s = s_1 + \cdots + s_n$, the set of possible values of $s_1, ..., s_n$ reside on a hyperplane in the Euclidian n-space, with the constraint that the points they determine must be in an n dimensional cube with side length S. Let R be the set of points in the intersection of the hyperplane with the cube. We can divide R in 2^{n-1} disjoint classes, such that points in a class all determine the same value of $(b_1, ..., b_n)$. In order to show that revealing s reveals very little information about $(b_1, ..., b_n)$, other than $b_1 \oplus ... \oplus b_n$, we only have to show that all 2^{n-1} classes have approximately the same size. From points in one class, we can get points in any other class by adding a vector with only $0, 1$ or -1 as coordinates. Clearly, the difference in size between two classes is upper bounded by the size of $(R + a) \setminus R$, where a is the vector added. Since a is "short", this size is in turn upper bounded by E, the size of the edge of R. This implies that

$$|Prob((b_1, ..., b_n) = (x_1, ..., x_n) \mid s = y) -$$

$$Prob((b_1, ..., b_n) = (x'_1, ..., x'_n) \mid s = y)| < E/C$$

where C is the size of the smallest class. There is at least one class larger than $|R|/2^{n-1}$, and hence $C > |R|/2^{n-1} - 2^{n-1}E$. It is relatively easy to see that R is a regular polytope, and that therefore $|R|$ is larger than E by a factor roughly equal to the diameter of R. So it is enough to show that this diameter can be expected to be exponentially large in $log(p)$, since then by choosing $log(p)$ polynomial in n, we can make 2^{n-1} be negligible compared to the diameter. Recall that the prover chooses all the s_i's uniformly between 0 and S. Clearly, all of them will be in the interval from $\sqrt{S}$ to $S - \sqrt{S}$ with probability exponentially close to 1. Moreover, if this occurs, the diameter of R is guaranteed to be at least $\sqrt{S}$.□

The reason why this is not quite enough is that if our blobs are to be used in a protocol for proving satisfiability of a circuit, it may be that the product from above is not opened, but is to be further multiplied with other blobs. In a circuit of large depth, this could lead to large s values, which by section 2 cannot be tolerated because it may allow the prover to cheat.

We therefore propose the following to solve the problem: we introduce another bound $S' > S$. Whenever a blob B could contain an s-value larger than S', two extra blobs are introduced. One of these, B'', contains the same bit as B, and the other one, B', is a random blob. Both are required to have s-value less than S, and after this point, B'' is used in place of B on the given wire of the circuit.

The verifier randomly chooses either to ask the prover to open B', or to ask the prover to open $B'B$. In both cases, the prover also opens $B''B$, showing that the XOR of B and B'' is zero.

The verifier will stop, if the prover does not open correctly all the blobs he is asked for. If the protocol is repeated k times, then with probability at most 2^{-k} will the prover be able to survive this protocol with a B'' for which he does not know as s-value less than $2S$ and larger than $-S$. Moreover, by Theorem 1, the protocol will only release an exponentially small amount of additional information about the contents of B''.

We remark that if our blobs are to be used in the interactive arguments of [4], this check protocol will often not be necessary: whenever a blob is used as input to a standard interactive gate, the overall protocol includes either the possibility that the blob be opened, or that it is multiplied with a blob B' and that either the product or B' is opened. Thus in this context, the check protocol only has to be applied in two cases:

- If an input blob does not go into an interactive gate (in this case B and B'' are the same blob and the prover need not open $B''B$).
- If a long series of XOR's is performed, with no intermediate interactive gates.

Note that even in the cases where the check does have to be done, this will not cause more rounds in the protocol, since we can use the verifier's challenge in the overall construction as challenges in the check protocol(s) as well. Thus they can be executed "in parallel" with the rest of the proof.

4 Remarks

We note that with these blobs, by Theorem 1, obtaining contradictory XOR-certificates (proofs that two blobs are encryptions of the same or different bits) constitutes breaking the system (allowing the prover to create a blob which she can decode as both a zero-blob and a one-blob). The XOR-certificates are not transitive, however, so these blobs do not have the strong equality property (see [2] or [3]). It is possible, for example, for the prover to create three blobs, none of which she can actually open, for which she can prove contradictory relations, namely that all of the blobs are different from each other pairwise (hence checking that the prover can open his input blobs to a circuit is really necessary).

The prover could do this as follows. Call the three blobs B_1, B_2, and B_3. The prover could first select the s- and r-values of the results randomly, so $B_1B_2 = a^{s_1}g^{r_1}$, $B_2B_3 = a^{s_2}g^{r_2}$, $B_1B_3 = a^{s_3}g^{r_3}$, where s_1, s_2, and s_3 are odd and the sum of r_1, r_2, and r_3 is odd. Then setting B_1 to one of the square roots of $a^{s_1-s_2+s_3}g^{r_1-r_2+r_3}$, B_2 to one of the square roots of $a^{s_1+s_2-s_3}g^{r_1+r_2-r_3}$, B_3 to one of the square roots of $a^{-s_1+s_2+s_3}g^{-r_1+r_2+r_3}$. Then all of the products will work correctly even though the prover is trying prove that an inconsistent set of relations is satisfiable.

This example requires the prover to take square roots of blobs. Thus it is natural to speculate that the problem could be avoided by demanding that all blobs be quadratic non-residues. Unfortunately, it is also possible to have an inconsistent set of relations which leads to the prover needing to take other roots than just square roots (see Appendix A).

This problem appears to be difficult to avoid. Rather than simply multiplying blobs together when computing the XOR, the verifier could randomly choose for each blob whether or not to use the blob or its multiplicative inverse in this product. Then the verifier could tell the prover which choices he has made. This would work perfectly fine except that, if there are any blobs which the prover would never have to open or prove equality to blobs that are opened, these blobs could have arbitrarily large s-values, possibly allowing the prover to cheat.

These blobs are, however *normal* (see [2] or [3]), so they will work in any protocol which only requires that the blobs have the standard properties, plus this one.

Appendix A.

Let $B_1, B_2, ..., B_5$ be blobs representing the bits $b_1, b_2, ..., b_5$. Consider the following inconsistent set of relations:

$$\begin{aligned}
XOR(b_1, b_2) &= 1\\
XOR(b_1, b_3) &= 1\\
XOR(b_2, b_4) &= 0\\
XOR(b_3, b_5) &= 1\\
XOR(b_1, b_4, b_5) &= 1\\
XOR(b_2, b_3, b_5) &= 1
\end{aligned}$$

In order to cheat and convince the verifier that these were satisfiable, the prover could first select the s- and r-values of the results randomly, so $B_1B_2 = a^{s_1}g^{r_1}$, $B_1B_3 = a^{s_2}g^{r_2}$, $B_2B_4 = a^{s_3}g^{r_3}$, $B_3B_5 = a^{s_4}g^{r_4}$, $B_1B_4B_5 = a^{s_5}g^{r_5}$, $B_2B_3B_5 = a^{s_6}g^{r_6}$, where s_1, s_2, s_4, and s_5 are odd, but s_3 is even. The value s_6 should be $2s_1 - s_2 + s_3 + 4s_4 - s_5$ and r_6 should be $2r_1 - r_2 + r_3 + 4r_4 - r_5$. If $p-1$ is not divisible by 3, then any random values will do for the first five r-values; otherwise, the r-values can be chosen randomly subject to the restriction that every product $a^{s_i}g^{r_i}$, for $1 \leq i \leq 5$, is a cube.

To find blobs that satisfy this, the prover can simply take cube roots to obtain $B_1, B_2, ..., B_5$ from the following congruences:

$$\begin{aligned}
B_1^3 &= a^{s_1+s_2-s_3-s_4+s_5}g^{r_1+r_2-r_3-r_4+r_5}\\
B_2^3 &= a^{2s_1-s_2+s_3+s_4-s_5}g^{2r_1-r_2+r_3+r_4-r_5}\\
B_3^3 &= a^{-s_1+2s_2+s_3+s_4-s_5}g^{-r_1+2r_2+r_3+r_4-r_5}\\
B_4^3 &= a^{-2s_1+s_2+2s_3-s_4+s_5}g^{-2r_1+r_2+2r_3-r_4+r_5}\\
B_5^3 &= a^{s_1-2s_2-s_3+2s_4+s_5}g^{r_1-2r_2-r_3+2r_4+r_5}
\end{aligned}$$

References

[1] J. Boyar, M. Krentel, and S. Kurtz. A discrete logarithm implementation of zero-knowledge blobs. Technical Report 87-002, University of Chicago, 1987. To appear in Journal of Cryptology.

[2] J. Boyar, C. Lund, and R. Peralta. On the communication complexity of zero-knowledge proofs. Submitted for publication.

[3] J. Boyar and R. Peralta. On the concrete complexity of zero-knowledge proofs. In *Advances in Cryptology - proceedings of CRYPTO 89*, Lecture Notes in Computer Science, pages 507–525. Springer-Verlag, 1990.

[4] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37:156–189, 1988.

[5] G. Brassard and C. Crépeau. Nontransitive transfer of confidence: a perfect zero-knowledge interactive protocol for SAT and beyond. In *Proceedings of the 27th IEEE Symposium on the Foundations of Computer Science*, pages 188–195, 1986.

[6] D. Chaum, I. Damgård, and J. van de Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *Advances in Cryptology - proceedings of CRYPTO 87*, Lecture Notes in Computer Science, pages 87–119. Springer-Verlag, 1988.