On the concrete complexity of zero-knowledge proofs

Joan Boyar*
Computer Science Department
University of Chicago

René Peralta†
Computer Science Department
University of Wisconsin - Milwaukee

Abstract

The fact that there are zero-knowledge proofs for all languages in NP has, potentially, enormous implications to cryptography. For cryptographers, the issue is no longer "which languages in NP have zero-knowledge proofs" but rather "which languages in NP have practical zero-knowledge proofs". Thus, the concrete complexity of zero-knowledge proofs for different languages must be established.

In this paper, we study the concrete complexity of the known general methods for constructing zero-knowledge proofs. We establish that circuit-based methods have the potential of producing proofs which can be used in practice. Then we introduce several techniques which greatly reduce the concrete complexity of circuit-based proofs. In order to show that our protocols yield proofs of knowledge, we show how to extend the Feige-Fiat-Shamir definition for proofs of knowledge to the model of Brassard-Chaum-Crépeau. Finally, we present techniques for improving the efficiency of protocols which involve arithmetic computations, such as modular addition, subtraction, and multiplication, and greatest common divisor.

1 Introduction

From a practical point of view, the bottleneck in the zero-knowledge interactive proof systems and in the "interactive arguments"[1] which are produced by the techniques of [1, 5, 7, 14] is the amount of interaction necessary. In this paper we provide powerful

*Supported in part by NSA Grant Number MDA904-88-H-2006.

†Supported in part by NSF Grant Number CCR-8909657.

[1]Protocols, which are proofs in the model of Brassard-Chaum-Crépeau [4], are not technically interactive proof systems, so we will refer to them as *interactive arguments*.

G. Brassard (Ed.): Advances in Cryptology - CRYPTO '89, LNCS 435, pp. 507-525, 1990.

tools for reducing the number of bits communicated. We call the number of bits communicated the "communication cost" of the proof system.

2 The communication cost of the known general methods

Let L be a language which has a zero-knowledge interactive proof system or interactive argument. Fix a zero-knowledge proof for L. Let $CC_k(n)$ be the number of bits communicated in this zero-knowledge proof to achieve probability of error no more than $(1/2)^k$, for a security parameter k, when the input size is n.

The technique of GMW for producing a zero-knowledge interactive proof system for an arbitrary NP language proceeds by reduction of the language to the graph 3-colorability problem. A simple, zero-knowledge protocol is then used to prove that the resulting graph is 3-colorable. Thus we are interested in the communication cost of GMW's protocol for graph 3-colorability. Let n be the number of vertices of the graph and e be the number of edges. The GMW proof involves the encryptions of the colors of vertices. In this paper we will consider as constant the cost of encrypting one bit. Thus, in the graph 3-colorability problem, encrypting a color also has constant cost.[2] With this parameterization, the complexity turns out to be $CC_k^{3COL}(n) \in O(k \cdot e \cdot n)$.

If L is a language in NP, then a zero-knowledge proof system for L would be constructed as follows:

1. Transform L to a SAT instance S via Cook's theorem [10].
2. Transform S to a 3-SAT instance 3S via [10].
3. Transform 3S to a graph 3-colorability instance G via [13].
4. Prove, via GMW, that G is 3-colorable.

The size of the SAT instance S depends on the time complexity of the NDTM for L. Let us assume this time complexity to be linear in the size n of the problem instance. Then the transformation at step 1 will produce a SAT instance of size $O(n^2)$. With no further blowup at steps 2 and 3, the final graph will have $O(n^2)$ vertices and edges. This yields a complexity of the interactive proof $CC_k(n) \in O(k \cdot n^4)$. If the time complexity of the NDTM for L is $O(n^2)$, as it would be for many number theoretic languages, the complexity of the interactive proof would be $CC_k(n) \in O(k \cdot n^8)$. Thus we see that the technique of reduction to graph 3-colorability, elegant as it is, cannot reasonably be used in practice for arbitrary languages in NP.

An alternative technique for producing zero-knowledge proofs ([1, 4, 5, 6, 7]), proceeds by reduction to a verifying boolean circuit rather than to an NP-complete

[2] In fact, this cost should probably be k rather than constant, but we make this simplifying assumption to avoid multiplying all communication costs mentioned in this paper by the same factor, k.

language.[3] From now on we refer to this technique as "circuit-based proofs". In circuit-based proofs, the communication cost is $CC_k(S) \in O(k \cdot S)$, where S is the circuit size. The circuit size has been related to the time necessary for membership verification by a Turing Machine. If this time is $p(n)$ for a problem of size n, then there exists a verification circuit of size $O(p(n) \log p(n))$ ([20]). If the time complexity is linear, then the complexity of this protocol is $CC_k(n) \in O(k \cdot n \log n)$. If the time complexity is quadratic, then the complexity of this protocol is $CC_k(n) \in O(k \cdot n^2 \log n)$.

Thus the circuit-based methods have the potential of producing proofs which can be used in practice. However, if the known techniques are used, the hidden constants in the expressions for the asymptotic costs are quite large. In this paper, we introduce several techniques which greatly reduce the concrete complexity of circuit-based proofs.

3 Blobs

A blob encryption scheme is a technique for encrypting a single bit. Several implementations of blobs are described in [3, 4].

Formally, a blob encryption scheme is a function $\mathcal{E} : \{0,1\} \times D \rightarrow \Sigma^n$ for suitably large n and set D. Given $y \in \Sigma^n$ a blob decryption is a string $x \in D$ such that either $\mathcal{E}(0,x) = y$ (y is a 0-blob) or $\mathcal{E}(1,x) = y$ (y is a 1-blob).

The following properties of this bit encryption technique are relevant to this paper.

- **Security** - the blobs that encrypt zero have the same distribution as the blobs that encrypt one, so Peggy's[4] blobs contain no information, in the information-theoretic sense.
- **Authenticity and Independence** - after creating several blobs, Peggy can reveal the bit encrypted to produce one of those blobs, she can convince Vic that this was in fact the bit she encrypted to produce that blob, and she can do this without revealing any information about any of the other blobs.
- **Equality** - if Peggy produced two blobs from the same bit b, then she can convince Vic of this fact without revealing b or any other information. Similarly, she can convince Vic that two blobs are encryptions of different bits if that is the case.

Consider, for example, the following blob encryption scheme:

Definition 1 *(Brassard-Crépeau) - Let N be a fixed composite number and α be a fixed quadratic residue modulo N. We define "BC-blobs" by $\mathcal{E}(0,x) = x^2 \ (mod\ N)$ and $\mathcal{E}(1,x) = \alpha x^2 \ (mod\ N)$, for $x \in Z_N^*$.*

[3] Impagliazzo and Yung [17] also present a technique for giving direct proofs for the computation of a Turing machine.

[4] In this paper, it will at times be convenient to think of the verifier as being named Vic, and the prover being named Peggy. This has the advantage that personal pronouns such as "he" and "she" can be used to unambiguously identify one of the parties.

Thus, BC-blobs encrypt a 0 by the square of a random number in Z_N^* and a 1 by α times the square of a random number in Z_N^*. We note that protocols using BC-blobs begin with a zero-knowledge subprotocol whereby the verifier convinces the prover that α is a square modulo N and that he knows a square root of α. The fact that there is an exponentially small probability that the verifier can cheat during this subprotocol means that BC-blobs can only yield almost perfect (or statistical) zero-knowledge proofs.

The authenticity property of blob encryption functions (see [3]) implies that the prover can not produce $x_1, x_2 \in D$ such that $\mathcal{E}(0, x_1) = \mathcal{E}(1, x_2)$. BC-blobs rely on the assumption that it is impossible to compute a square root of α modulo N in probabilistic polynomial time. If the prover was to somehow produce x_1, x_2 such that $\mathcal{E}(0, x_1) = \mathcal{E}(1, x_2)$, then we have $x_1^2 = \alpha x_2^2$ which implies $\sqrt{\alpha} = (x_1/x_2)(mod\ N)$.

Definition 2 *We call the evidence that Peggy gives when convincing Vic that a blob is the encryption of a particular bit the "decryption" of the blob. We call the evidence Peggy gives to show that she produced two blobs from the same bit or from different bits the "XOR-certificate" of the two blobs. We assume that both blob decryptions and XOR-certificates are strings of bits (i.e. non-interactive proofs).*

Given BC-blobs u, v the prover convinces the verifier that u, v encrypt the same bit by displaying a square root of uv modulo N. The prover convinces the verifier that u, v encrypt different bits by displaying a square root of αuv modulo N. These square roots are the XOR-certificates of BC-blobs.

Let $t_1 = \mathcal{E}(b_1, x_1)$, $t_2 = \mathcal{E}(b_2, x_2)$, and $t_3 = \mathcal{E}(b_3, x_3)$. That is, t_i is a BC-blob encryption of b_i. The reader can easily verify that given XOR-certificates $C_{1,2}$ between t_1 and t_2 and $C_{2,3}$ between t_2 and t_3, an XOR-certificate for t_1 and t_3 is given by $C_{1,3} = (C_{1,2}C_{2,3}/(t_2\alpha))\ (mod\ N)$ if $b_1 \neq b_2$ and $b_2 \neq b_3$ or by $C_{1,3} = (C_{1,2}C_{2,3}/t_2)\ (mod\ N)$ otherwise. Thus, given a set of BC-blobs $\{t_i\}$ and XOR-certificates $\{C_{i,j}\}$ for some pairs of blobs, if the relationship between blobs t_r and t_s can be inferred from $\{C_{i,j}\}$ then the XOR-certificate $C_{r,s}$ is computable in polynomial time.

Definition 3 *When an XOR-certificate for blobs t_1, t_3 is computable in polynomial time from the XOR-certificates for blobs t_1, t_2 and t_2, t_3, we say that XOR-certificates are "transitive".*

We note that in BC-blobs, obtaining contradictory XOR-certificates constitutes breaking the system: if $C_{1,2}$ certifies that t_1 and t_2 encrypt the same value and $C'_{1,2}$ certifies that t_1 and t_2 encrypt different values, then $\sqrt{\alpha}$ is given by $\sqrt{\alpha} = (C_{1,2}C'_{1,2})/(t_1t_2)\ (mod\ N)$. This motivates the following definitions.

Definition 4 *We say that the prover "breaks" the blob encryption scheme $\mathcal{E}$ if it can compute x_1, x_2 such that $\mathcal{E}(0, x_1) = \mathcal{E}(1, x_2)$.*

Notice that our definition is very strong in the sense that the ability to decrypt a single blob into two different values constitutes breaking the blob encryption scheme.

Definition 5 *An implementation of blobs has the "strong equality property" if and only if (1) the XOR-certificates are transitive and (2) obtaining contradictory XOR-certificates constitutes breaking the system.*

Finally, we note that several implementations of blobs in the literature, including BC-blobs and those based on the discrete logarithm problem (see [3, 8]), are "normal" in the sense of the following definition.

Definition 6 *A blob encryption scheme $\mathcal{E}$ is "normal" if there is a BPP machine T, such that from any pair of blobs t_1 and t_2 created by $\mathcal{E}$, any decryption x_1 of t_1, and any XOR-certificate $C_{1,2}$ for t_1, t_2, T can compute a valid decryption x_2 of t_2.*

Normality will turn out to be an important property in proving that our protocols are proofs of knowledge.

4 The Brassard-Crépeau proof system

Let us recall the Brassard–Crépeau interactive arguments. The prover and verifier, Peggy and Vic, have agreed upon a circuit which has S gates, all of which have bounded (or at most $O(\log S)$) fan-in; they have agreed upon a security parameter k; Peggy knows a satisfying assignment for the circuit; and she wishes to convince Vic that she knows such an assignment. In order to do this, Peggy, who is assumed to have only BPP power, uses a blob encryption scheme. The protocol is as follows:

Peggy and Vic repeat the following steps k times.

1. For each gate in the circuit, Peggy produces a truth table for the boolean function computed by that gate. For each of the truth tables, she then, randomly and independently, chooses a permutation of the rows. For every wire in the circuit, she determines which value that wire would be carrying if the input wires carried the values corresponding to her satisfying assignment. Then, Peggy encrypts all of the bits on the wires and all of the bits in her permuted truth tables and sends the resulting blobs over to Vic.

2. Vic then flips a coin to get a random bit b and sends that bit over to Peggy.

3. If $b = 0$, Peggy shows Vic that all of the truth tables were formed correctly by decrypting them. She will also show that the blob for the output of the circuit is an encryption of a one. If $b = 1$, Peggy shows that the inputs and output of each gate correspond to some row in the truth table for that gate. She does this using the blob equality property, showing that the blobs, on the wires which are the inputs and output of the gate, are encryptions of the same bits as the blobs in some row of the truth table.

4. Vic checks everything that Peggy has sent him.

5 Eliminating truth tables

In this section, we will present techniques which can be useful in improving the efficiency of perfect and almost perfect zero-knowledge proofs which use the techniques of the original Brassard–Crépeau [5] proof system.[5]

In designing circuits for NP-problems, one would often want to use MAJORITY gates or AND or OR gates with large fan-in. It is, of course, relatively easy to replace these gates with AND, OR, and NOT gates with bounded fan-in so that one could directly apply the Brassard–Crépeau [5] proof system. It is more efficient, however, to skip this replacement and to use the techniques in the following sections to prove that these gates with large fan-in work correctly. These techniques can also be used to improve the efficiency of the protocols for circuits which only have gates with bounded fan-in.

5.1 MAJORITY gates

First, consider MAJORITY gates with fan-in n, where $n = 2k+1$. The output should be one if at least $k+1$ of the inputs are one, and zero if at least $k+1$ of the inputs are zero. Thus, we need only show that there are $k+1$ inputs which are equal to the output. In order to hide which of the inputs are the same as the output, we will have Peggy produce n more blobs corresponding to the original input blobs for this gate. These additional n blobs will be encryptions of the same bits as the input blobs, but Peggy will send them to Vic in random order, so Vic will be unable to determine the correspondence. Then if the bit b Vic sends to Peggy is a zero, Peggy will show the correspondence between the input blobs and the n additional blobs, using the equality property. If the bit b is a one, Peggy will show that $k+1$ of the additional blobs are encryptions of the same bit as the output bit, again using the equality property. If the number of inputs equal to the output is greater than $k+1$, Peggy will randomly choose $k+1$ additional blobs which have the same value as the output.

MAJORITY gates can be used to simulate AND and OR gates. Consider first an AND gate with n inputs. This can be simulated by a MAJORITY gate with $2n-1$ inputs, n of which are the same inputs as to the AND gate and $n-1$ of which carry the value zero. The output of this majority gate will be one if and only if all of the inputs to the AND gate were one, so the simulation is correct. OR gates with n inputs can be simulated similarly, but in this case the extra $n-1$ inputs must all be ones. Thus, in order to simulate an AND or OR gate with n inputs, Peggy will create $2n-1$ additional blobs, with n corresponding to the inputs to the AND or OR gate and $n-1$ being encryptions of zeros for an AND gate or ones for an OR gate. Then, if the bit b Vic sends to Peggy is a zero, Peggy will show the correspondence between the inputs to the AND or OR gate and n of the additional blobs, and she will decode the remaining $n-1$ blobs to show that they are all zeros or all ones, depending on

[5] Unfortunately the techniques presented here must be greatly modified for compatibility with the complementation of the bits on the wires, which occurs with the Brassard–Chaum–Crépeau system [4] .

whether the gate was an AND or OR gate. If the bit b is a one, Peggy will show that n of the additional blobs are encryptions of the same bit as the output bit.

NOT gates require no additional blobs. When Vic sends Peggy $b = 0$, Peggy will show that the NOT gates are working correctly by using the equality property to show that the blobs on the inputs and outputs of the NOT gates are encryptions of different bits. Peggy can ignore the NOT gates when $b = 1$.

Notice that the simulation of AND and OR gates with large fan-in never required that the fan-in be greater than two. If the original gates had only two inputs, Peggy would only need to create three additional blobs per gate, rather than the twelve needed to represent a truth table.[6] If the number of inputs is N and the number of gates is S, the total number of blobs on the circuit wires is $N+S$. If T of the S gates are NOT gates, then the original protocols which use truth tables use $N + 13(S - T) + 5T$ blobs per round, while the protocols described here use $N + 4(S - T) + T$ blobs per round, at most $(N + 4S)/(N + 13S)$ times as many. This is approximately three times more efficient than the Brassard-Crépeau system. Of course, the gain in efficiency is much greater when the circuit contains MAJORITY gates with large fan-in.

5.2 PARITY gates

Peggy can show that PARITY gates with n inputs are working correctly using $n + 1$ additional blobs. These blobs will correspond to the n inputs plus the output. When $b = 0$, Peggy will show the correspondence. When $b = 1$, Peggy can show that a PARITY gate is working correctly by exhibiting a pairing of the additional blobs for the gate (including the blob for the output bit), and using the equality property to show that the blobs within a pair are encryptions of the same bit. If n is even, there will be an unpaired additional blob, and Peggy must show that it was the encryption of a zero.

5.3 Proofs of security

For simplicity we assume, throughout this section, that no two NOT gates have the same input, that the output of one NOT gate is never the input to another NOT gate, and that the output of every gate (except for one) is an input to some other gate. We impose no further restrictions on fan-out.

We begin by showing that our interactive arguments are perfect or almost-perfect zero-knowledge proofs, depending on which implementation of blobs is used. Note that our interactive arguments are obviously zero-knowledge if the blob implementation used has the forgeability property (see [3]). The following theorem shows that the forgeability property is not necessary.

Theorem 1 *Our interactive arguments are perfect zero-knowledge if Peggy and Vic*

[6] Independently of this work, den Boer[11] has also decreased the number of blobs necessary for validating the computation of binary boolean functions.

use any implementation of blobs such as that in [3, 8], which leads to perfect zero-knowledge proofs in the original Brassard-Crépeau interactive arguments.

proof : We will construct a simulator for these interactive arguments. The simulator starts by setting the bits on the wires so that the output of the circuit is equal to one and so that the input of every NOT gate is different from the output of the same gate. To do this, the simulator assigns values to the wires, level by level, starting with the inputs. First, the input wires are all given the value one. The first level of gates is the set of gates which have as their inputs the inputs to the circuit. The kth level of gates is the set of gates which are not on any of the first $k-1$ levels and which have as their inputs only the inputs to the circuit and the outputs of gates at lower levels. As the simulator is assigning values to the wires corresponding to the outputs of the gates at level k, it will negate the value on the input wire for all NOT gates, and it will assign the value one otherwise. The simulator has completed its first step unless the final gate in the final level, level t, of the circuit is a NOT gate and its input is a one. If this is the case, the input to this final gate came from a unique gate on level $t-1$. The simulator will change the output of that gate from a one to a zero. Since we have assumed that this gate on level $t-1$ cannot be a NOT gate, the simulator has finished its first step.

Then, the simulator flips a coin in an attempt to guess which bit Vic will send. If the coin flip leads to a guess of $b=0$, the simulator will set up the additional blobs for each gate so that they correctly correspond to the appropriate values on the input and output wires and so that those blobs which are supposed to have fixed values (for the AND and OR gates) have those values. The simulator then randomly permutes the additional blobs for each gate and sends everything over to Vic. If Vic sends $b=0$, the simulator will have no problem responding correctly. Otherwise it will back up the tape for the transcript it is creating and will try again.

If the simulator's coin flip leads to a guess of $b=1$, the simulator will set up the additional blobs for each gate so that it can respond correctly if $b=1$. For a MAJORITY gate with $n=2k+1$ inputs, it will set up all of the additional blobs with the same value as the output blob. For an n-input AND gate, it will set $n-1$ of the additional blobs to zero, and will make the remaining n blobs encryptions of the same bit which has been assigned to the output wire. An n-input OR would be similar except that the first $n-1$ additional blobs would be encryptions of one, rather than of zero. For a PARITY gate, all of the additional blobs can be encryptions of zero. Then, if Vic sends $b=1$, the simulator will have no problem responding correctly. Otherwise, it will back up the tape for the transcript it is creating and will try again. With each coin flip, the simulator has a 50-50 chance of guessing the correct value of b, so the simulation will take expected polynomial time.

The simulator will produce transcripts which have exactly the same distribution as those which would be produced with the true prover because of the blob properties mentioned above. The key to this is that zero blobs and one blobs are drawn from exactly the same distributions .□

Theorem 2 *If a blob implementation is used which leads to almost perfect zero-knowledge proofs in the original Brassard–Crépeau interactive arguments, then our interactive arguments are almost perfect zero-knowledge.*

proof : Analogous to the proof of the previous theorem .□

Now we will show that our interactive arguments are proofs of knowledge. We will use the formalism developed in [12], which we will call the "FFS model". In the FFS model the prover and verifier are BPP in power and have an input tape, a random tape, and a "knowledge" tape. Paraphrasing FFS, such a proof system is a proof of knowledge system for the predicate $P(I, W)$ [7] if

> There exists a polynomial-time probabilistic Turing machine M such that for any prover A (honest or dishonest), for any initial contents of A's knowledge tape K and random tape R, and any sufficiently large input I, if the execution of the protocol on input I (assuming an honest verifier B) succeeds with nonnegligible probability, then the output produced by M at the end of the execution of $M(A, R, K)$ on input I satisfies the predicate P with overwhelming probability. [8]

Given that our interactive arguments rely on the security of blob encryption mechanisms (and that the above definition allows arbitrary initialization of the prover's knowledge tape), the possibility arises that the prover can convince the verifier that the circuit is satisfiable simply because it "knows" how to break the blob encryption scheme. Our theorem will state that after the successful execution of the protocol the verifier has obtained a proof that either the prover knows a satisfying assignment to the boolean circuit or the prover knows how to break the blob encryption scheme. But what does it mean to "know" how to break a blob encryption scheme? As pointed out by [12], the concept of Turing machine "knowledge" is a very subtle one. Lest we are forced to develop a new formalism to capture this notion, we must state our definition (of what it means to know how to break a blob encryption scheme) in terms of (a BPP observer M) being able to compute a satisfying assignment to a predicate.

Definition 7 *We say that the prover "knows" how to break the blob encryption scheme $\mathcal{E}$ if it knows (in the sense of [12]) a satisfying assignment to the predicate $\mathcal{E}(0, x_1) = \mathcal{E}(1, x_2)$ where the free variables are x_1 and x_2.*

In order to show that a BPP observer M can break the blob encryption scheme if the prover cheats, we need to assume that the encryption scheme is normal (see section 3, definition 6).

---

[7] $P(I, W)$ should be thought of as a family of circuits indexed by I and with input a binary string W.

[8] For a formal definition see [12]. Note that our interactive arguments are "restricted input", not "unrestricted input", zero-knowledge proofs of knowledge.

Theorem 3 *Suppose the circuit contains AND, OR, NOT, and MAJORITY gates. After a successful execution of our protocol using a normal blob encryption scheme $\mathcal{E}$, the verifier is convinced that the prover knows either a satisfying assignment to the circuit or how to break the scheme $\mathcal{E}$. This remains so even if gates have unbounded fan-in.*

proof : From the proof in [12] that every problem in NP has an interactive proof system of knowledge which is zero-knowledge, we see that to prove that our protocol is a proof of knowledge, we need only show that if an observer is able to see Peggy respond to receiving $b = 0$ and $b = 1$ for the same set of blobs, that observer can learn a satisfying assignment for the circuit or a satisfying assignment to the predicate $\mathcal{E}(0, x_1) = \mathcal{E}(1, x_2)$. If the observer sees any prover respond correctly to both possible challenges for a given set of blobs, the observer can learn one of these satisfying assignments by performing a breadth-first search of the circuit, starting with the output, determining the values on the wires it encounters. When processing a gate G in this breadth-first search, the observer already knows the value of the output. If G is a NOT gate, the input is the complement of the output. If G is an AND, OR, or MAJORITY gate, the answer to the challenge $b = 0$ will tell the observer the correspondence between the additional blobs and the blobs on the inputs to G. The answer to the challenge $b = 1$ will show that some of the additional blobs are equal to the output blob. From the correspondence between the additional blobs and input blobs, the observer will be able to determine the values of enough of the inputs to force that output value. After completing the breadth-first search, the observer will know values for enough of the circuit's inputs to force an output value of one. The other inputs can be assigned arbitrary values. This process can fail if, in processing a gate, the observer gets a contradictory assignment to an input wire. That is, a wire which has already been assigned a boolean value c is now assigned $\bar{c}$. In this case, using the fact that the encryption scheme is normal, the observer can decrypt a blob both to a 1 and to a 0, and therefore can satisfy the predicate $\mathcal{E}(0, x_1) = \mathcal{E}(1, x_2)$.□

Now let us suppose the circuits also have PARITY gates which are treated as described in section 5.2. We will prove that if an implementation of blobs is used which has the strong equality property (see section 3, definition 5), then our protocols are still proofs of knowledge. Unfortunately, the blob implementation described in [3, 8], which leads to perfect zero-knowledge proofs does not have the strong equality property. The problem with this implementation is that it is possible for Peggy to create two blobs t_1 and t_2 which she is unable to decrypt as either a zero or a one but which she can "prove" are both encryptions of the same bit and encryptions of different bits.[9]

Suppose we attempt the breadth-first search technique used in the proof of the previous theorem in a circuit which contains PARITY gates. When the observer

[9]She can do this by setting $t_1 = g^{r_1}\sqrt{g^c a} \pmod p$ and $t_2 = g^{r_2}\sqrt{g^c a} \pmod p$, where c is 0 or 1 depending on whether a is a quadratic residue or nonresidue, respectively.

encounters a PARITY gate, he may not learn enough of the gate's input values to force the correct output. If the PARITY gate has an even number of inputs, the observer will learn that the value on some wire is a zero and it will learn which wire, though that wire might be the gate's output. If the output of the PARITY gate is paired with one of the inputs, the observer will learn that that particular input wire carries the same value as the output. For the other wires, the observer will only learn that some wires carry the same value as some other wires. We will show, however, that this is enough information for the observer to efficiently compute a satisfying assignment.

Theorem 4 *Suppose the circuit contains AND, OR, NOT, MAJORITY, and PARITY gates. Suppose that the blob implementation used has the strong equality property. Then, after a successful execution of our protocol the verifier is convinced that the prover knows either a satisfying assignment to the circuit or how to break the blob encryption scheme. This remains so even if gates have unbounded fan-in.*

proof : As in the proof of the previous theorem, we assume that the observer has seen the prover respond to both a $b = 0$ challenge and a $b = 1$ challenge from the verifier and we perform a breadth-first search of the circuit.

Before beginning the breadth-first search, the observer should assign distinct symbolic variables to the wires of the circuit. The symbolic variable on a wire which is the output of a NOT gate can be pushed through the gate by assigning the negation of that symbolic variable to the input. Thus the wires may be assigned a symbolic variable, the negation of a symbolic variable, or a Boolean value. In the breadth-first search, whenever the observer learns that two wires carry the same value, either because of a PARITY gate or because one is the input wire to an AND, OR, or MAJORITY gate and one is the output wire, both wires should be given the same value, according to the following rules:

- If both wires have a Boolean value, they should have the same value, so no updating is necessary. If they have different values, the prover has cheated; the observer should terminate the breadth-first search and break the system as described below.

- If one wire has a Boolean value and the other has a symbolic variable, all wires in the circuit labeled with that symbolic variable should be given the Boolean value and all wires labeled with the negation of that symbolic variable should be given the complement of that Boolean value.

- Similarly, if one wire is labeled with a Boolean value and the other has the negation of a symbolic variable, all wires in the circuit labeled with that symbolic variable or its negation should be labeled with the appropriate Boolean value.

- If both wires are labeled with symbolic variables or both are labeled with the negations of symbolic variables, all wires currently labeled with one of them or with the negation of that one should be labeled with the other or the negation of the other.

- If one wire is labeled with a symbolic variable and the other with the negation of a second symbolic variable, those wires labeled with the second symbolic variable should be relabeled with the negation of the first symbolic variable, and those wires labeled with the negation of the second symbolic variable should be relabeled with the first symbolic variable.

After the breadth-first search has been completed, some input wires will have been assigned Boolean values, but some may still be labeled with symbolic variables or the complements of symbolic variables. The observer can then assign arbitrary Boolean values to the symbolic variables and let these assignments determine values for those input wires which have not yet been assigned Boolean values. The resulting assignment will force the output of the entire circuit to be a one.

To see this, consider numbering the gates according to the order in which they are processed during the breadth-first search, with the output gate being given the number one. Look at the subcircuit C_k defined by the first $k-1$ gates. The inputs to this subcircuit are all encountered before we begin processing the kth gate. We claim that if these inputs to C_k are given any Boolean values consistent with the labels on the corresponding wires immediately before the processing of gate k, then the output of the circuit will be a one. This can be proved by induction on k. The base case, $k = 1$, is easy because before the first gate is processed, the output is given the value one. Suppose this is true for k and let us look at the kth gate. The processing described above ensures that the inputs to this gate will get appropriate values to force the output value. This processing may affect the labeling of previously encountered wires, but only to be more restrictive in the values allowed. If the labeling becomes so restrictive that no Boolean values are consistent with the labeling, then the prover must have cheated either in decrypting some blobs or in showing that two blobs are equal or not equal to each other. In this case, the observer can break the encryption scheme as described below. Thus, if the output for the subcircuit C_k is one for any consistent (according to the labeling immediately before gate k is processed) set of Boolean values assigned to the inputs, the output for the subcircuit C_{k+1} will be one if the inputs to C_{k+1} are given any Boolean values consistent with the labels on the corresponding wires after the processing of gate k. Since C_S is the entire circuit, the method described above will enable the observer to find a satisfying assignment, unless the observer found a contradiction.

If such a contradiction occurs, the observer can break the blob encryption scheme. Any contradiction which arises results from two strings of blob equalities and blob inequalities, with both strings starting and ending at the same blobs. Since each blob equality or inequality is proven with an XOR-certificate, and since these certificates are transitive, the observer has a certificate that the first blob on these strings is the encryption of the same bit as the last blob and also a certificate showing that they are encryptions of different bits. Since the blob implementation has the strong equality property, the observer can use these contradictory certificates to break the blob encryption scheme .□

6 Integer operations

Convincing an adversary that a circuit is satisfied by an input with a given blob encryption is a special case of the following problem:

> Suppose that a given circuit computes a function $F(I)$. Then, given the blob encryption of an input I, and a blob encryption of an output O, convince an adversary that $F(I) = O$.

There are several alternatives to the straightforward verification of the circuit's computation. The protocol for verifying $F(I) = O$ could be, for example, probabilistic in the sense that, even if the prover is honest, the protocol only shows that $F(I) = O$ with probability exponentially close to one. Another technique is to express the function F as a composition of two functions $F = G \circ H$. In this case, the verifier could produce a blob encryption I' of $H(I)$ and convince the verifier that $H(I) = I'$ and $G(I') = O$. If the H and G are chosen appropriately then the communication cost of proving the latter two statements may be smaller than the communication cost of the direct proof (and, of course, it would still be a zero-knowledge proof).

A third technique is to add to the input I whatever additional information I' (where I' will also be encrypted using blobs) might help establish $F(I) = O$. More precisely, we seek a relation $R(I', I, O)$ such that $F(I) = O$ if and only if there exists an I' such that $R(I', I, O)$ holds. For example, if $F(I)$ is the GCD function of two integers A and B, then we let $R(X, Y, A, B, F(A, B)) = [(X * A + Y * B) = F(A, B)]\&[F(A, B)$ *divides* $A]\&[F(A, B)$ *divides* $B]$ (here $I = (A, B)$ and $I' = (X, Y)$). Then $F(A, B) = O$ if and only if there exists $I' = (X, Y)$ such that $R(X, Y, A, B, O)$ holds. This technique is useful when the communication cost of proving $R(I', I, O)$ is smaller than the communication cost of directly proving that $F(I) = O$. We note that this technique is closely related to the concept of program "checking", as developed by Blum and Kannan [2] and therefore their methods may prove useful in lowering the communication complexity of zero-knowledge proofs.

It is outside the scope of this paper to classify and explore all the different techniques that might prove useful in lowering the communication cost of zero-knowledge proofs. We focus on integer multiplication/division and GCD computation, which are common operations requiring circuits which are super-linear in size. We make use of the techniques discussed above to design protocols for these operations which have almost-linear communication cost.

For integer operations, we formulate the problem as follows : Let $OP(X, Y)$ be a binary operation on integers. Let X, Y, Z be integers such that $OP(X, Y) = Z$. Let $F(X), F(Y), F(Z)$ be blob-based encryptions of X, Y and Z respectively. Then we want a zero-knowledge protocol by which the prover can convince the verifier that $OP(X, Y) = Z$.

We start by noting that the techniques of section 3 reduce the communication cost of proofs for integer addition by a factor of seven. To see this, note that a circuit for adding two n-bit numbers requires 5 gates per bit (2 XOR gates, 2 AND gates, and 1 OR gate). Therefore the BC protocol requires the prover to produce $65 \cdot n$ additional blobs. By using PARITY and MAJORITY gates instead, it is not hard to see that

only $9 \cdot n$ additional blobs are needed. This is an important optimization since integer addition is a basic step in circuits for most arithmetic operations. Subtraction, for instance, can be done using addition because convincing the verifier that $X - Y = Z$ is equivalent to convincing the verifier that $Z + Y = X$.

6.1 Multiplication

Let $F(A)$, $F(B)$, and $F(C)$ be given, where F is a blob-based encryption function. We will show a zero-knowledge protocol through which the prover can convince the verifier that $C = A \cdot B$ with $O(n \log n)$ communication cost when C is n bits long. This is much faster than the BCC and BC protocols would be if implemented in the obvious manner.

Note that the bit-length n of C is public, since blob-based encryption reveals length. Let p_i be the i^{th} prime and let m be the smallest integer such that $\prod_{i=1}^{m} p_i > 2^n > C$. Let $T_n = \{p_i : i = 1, \ldots, \lambda \cdot m\}$ where λ is a constant greater than 1. Note that the product of any m primes in T_n exceeds C. If $A \cdot B$ does not equal C, then $A \cdot B \equiv C \pmod{p_i}$ for at most $m - 1$ primes p_i. Thus $A \cdot B \not\equiv C \pmod{p_i}$ for at least $\lambda m - m + 1 > (\lambda - 1) \cdot m$ primes p_i. Thus, if a random $p_i \in T_n$ is chosen by the verifier then, with probability at least $1 - (1/\lambda)$, the prover will not be able to show that $A \cdot B \equiv C \bmod p_i$. This is the basis for our protocol.

Protocol for multiplication

1. The prover commits to $F(A)$, $F(B)$ and $F(C)$, where C is n bits long.
2. The verifier chooses a random prime $p \in T_n$.
3. The prover produces $(F(K_A), F(K_B), F(K_C), F(R_A), F(R_B), F(R_C))$ such that $0 \leq R_A, R_B, R_C < p$ and proves, using the BCC general protocol, that
   - $K_A \cdot p + R_A = A$
   - $K_B \cdot p + R_B = B$
   - $K_C \cdot p + R_C = C$
   - $R_A \cdot R_B = R_C \bmod p$.

Note that the 4 equations imply $A \cdot B \equiv C \bmod p$.

To show that $R_A \cdot R_B = R_C \bmod p$, the prover can commit to $F(k)$ and prove that $R_A \cdot R_B = R_C + k \cdot p$. The equalities can be shown using the equality property of blobs.

The communication cost of this protocol depends on the size of $p_{\lambda \cdot m}$, the largest prime in T_n. In order to obtain a concrete upper bound on $p_{\lambda \cdot m}$ we will need the following lemma:

Lemma 1 *For $x > 29$, the product of primes less than x is greater than 2^x.*

proof : We will use the following inequality (due to Chebyshev [9])

$$.92(x/\ ln\ x) < \pi(x) < 1.11(x/\ ln\ x).$$

Consider the sum

$$S(x) = \sum_{p<x} ln\ p = \sum_{p\leq x} ln\ p - \tau(x) ln\ x$$

where p denotes a prime number and $\tau(x)$ is 1 if x is prime and 0 otherwise. Using Theorem 6.15, page 145 of [19], it is not hard to show that

$$S(x) = \pi(x)\ ln\ x - \int_2^x \pi(t)/t\ dt - \tau(x) ln\ x.$$

Thus $S(x) > f(x)$ where

$$f(x) = .92x - 1.11 \int_2^x \frac{dt}{ln\ t} - ln\ x.$$

Notice that $f'(x) = .92 - 1.11 \cdot ln^{-1}(x) - x^{-1}$. Thus, for $x > 600$ we have $f'(x) > f'(600) > ln\ 2$. Therefore $f(x)$ increases faster than the line $(ln\ 2) \cdot x$ for $x > 600$. On the other hand, numerical integration reveals that $f(600) > 416 > (ln\ 2) \cdot 600$. Thus we have, for $x > 600$,

$$\prod_{p<x} p = e^{S(x)} > e^{f(x)} > e^{(ln\ 2)x} = 2^x.$$

Thus we have reduced the problem to that of verifying that $\prod_{p<x} p > 2^x$ for $29 < x \leq 600$. This verification was performed by computer .□

Corollary 1 *If $p_m > 29$, then $\log_2 C > p_m$.*

proof : Note that $C > \prod_{p<p_m} p$.

Corollary 2 *If $p_m > 29$, then $m < 2 \cdot (\log_2 C)/(\log_2 \log_2 C)$.*

proof : Since $\log_2 C > p_m$, we have $m < \pi(\log_2 C)$ and, using Chebyshev's inequalities, $\pi(\log_2 C) < 2 \cdot (\log_2 C)/(\log_2 \log_2 C)$.□

Corollary 3 *If $\lambda < (1/2) log_2 log_2 C$, then $p_{\lambda m} < 2\lambda \log_2 C$.*

proof : For $p_r > 5$ we have $p_r < r \log_2 r$ (this can be derived from theorem 3 of [21]). Therefore

$$\begin{aligned} p_{\lambda \cdot m} &< \lambda \cdot m \log_2(\lambda \cdot m) \\ &< \frac{2\lambda \log_2 C}{\log_2 \log_2 C}(\log_2 \lambda + 1 + \log_2 \log_2 C - \log_2 \log_2 \log_2 C) \\ &< 2\lambda \log_2 C \end{aligned}$$

where the second inequality follows from the previous corollary and the third inequality from the assumption that $\lambda < (1/2)log_2 log_2 C$.□

Thus, $p_{\lambda \cdot m}$ is no more than $\log_2(2\lambda n)$ bits long, where n is the size of C. Noting that the general BCC protocol has $O(r \cdot s)$ communication cost per round to multiply an r-bit number by an s-bit number, using the "classic" multiplication algorithm, we conclude that the communication cost of each round of our protocol is $O(n \cdot \log_2 n)$.

6.2 Integer GCD

Given blob-based encryptions $F(A), F(B), F(\delta)$ of integers A, B, δ, the prover wants to convince the verifier that $GCD(A, B) = \delta$. The communication cost of the BC and BCC proofs for this problem is $O(n^2)$ when A, B are n-bit integers. A much cheaper protocol is the following:

Protocol for integer GCD

1. The prover commits to $F(u), F(v), F(w)$ and $F(x)$ where $A \cdot u + B \cdot v = \delta$.
2. The prover proves
   - $\delta \cdot w = A$
   - $\delta \cdot x = B$
   - $A \cdot u + B \cdot v = \delta$.

Note that the 3 equations imply that $GCD(A, B) = \delta$.

The proofs at step 2 are done using the techniques of the previous section. Thus, the per-round communication cost of this protocol is $O(n \log n)$ for A and B of size n.

7 Combining logical gates and arithmetic gates

The techniques of sections 5 and 6 can be used to prove the satisfiability of circuits containing both logical and arithmetic gates. We first note that GCD, DIV, and MOD gates can be replaced by multiplication and addition gates (plus additional, prover supplied, inputs) in the manner of section 6.2. Since addition gates are implemented using standard (linear size) logical circuits, we need only consider a circuit containing logical gates and multiplication gates. Such a circuit can be proven satisfiable through the following protocol:

Protocol for circuits containing multiplication gates

1. The prover commits to blobs for each wire of the circuit (including the inputs and outputs of the multiplication gates) and to the additional blobs required by logical gates as described in section 5.

2. For each multiplication gate with inputs A, B and output C, the verifier sends a random prime p of size $2\lambda \log n$ where n is the size of the output to the gate.

3. The prover commits to blobs for the wires and to the additional blobs required by the logical gates in the circuit induced by step 3 in our protocol for multiplication (section 6.1). i.e. the boolean circuit induced by the statements

$$K_A \cdot p + R_A = A$$

$$K_B \cdot p + R_B = B$$

$$K_C \cdot p + R_C = C$$

$$R_A \cdot R_B = R_C + k \cdot p.$$

4. The verifier challenges the prover for either a proof that all of the blobs were constructed according to protocol (type 0 challenge) or for a proof that the output blob for each logical gate is correct (type 1 challenge).

5. The prover responds to the challenge.

If the circuit is not satisfiable then the commitment to the output blob for at least one gate is not the correct computation for that gate. If that gate is a logical gate the prover will be caught with probability (1/2). If the gate is a multiplication gate with inputs A, B and (incorrect) output C, then the prover will get caught if $A * B \neq C \bmod p$ (this happens with probability at least $1 - \frac{1}{\lambda}$) and the verifier sends the correct challenge. Thus the probability of cheating on each iteration is no more than $(1/2) + \frac{1}{2\lambda}$.

8 Remarks

The results in this paper have been significantly improved recently. Gilles Brassard has succeeded in reducing the communication cost of integer multiplication to $O(nc^{\log^* n})$, for a constant c. In addition, if BC-blobs are used, then the blobs for the outputs of NOT and PARITY gates can be computed by the verifier without help from the prover (and hence without communication) [5]. Carsten Lund has shown that the resulting protocol is still a restricted input zero-knowledge proof of knowledge.

The techniques described in this paper can also be used in circuit-based proofs which are interactive proof systems according to the GMR [16] definition. If one substitutes probabilistic encryption with the quadratic residuosity assumption [15] for the BC-blobs, everything still works (except that of course the proofs will now be only computational, rather than almost perfect, zero-knowledge).

At this conference, a paper by Kilian, Micali, and Ostrovsky [18] was presented which introduces a different technique for reducing the communication cost of zero-knowledge proofs. The techniques they use to reduce the number of envelopes necessary in "subset-revealing" protocols can easily be extended to protocols such as ours

which also reveal the XOR of pairs of bits. We believe combining their techniques and ours carries us a long way towards the goal of being able to produce zero-knowledge proofs which are practical for any NP problem.

References

[1] J. C. Benaloh. Cryptographic capsules: A disjunctive primitive for interactive protocols. In *Advances in Cryptology - proceedings of CRYPTO 86*, Lecture Notes in Computer Science, pages 213–222. Springer-Verlag, 1987.

[2] M. Blum and S. Kannan. Designing programs that check their work. *Proceedings of the 21th Annual ACM Symposium on the Theory of Computing*, pages 86–97, 1989.

[3] 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.

[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] G. Brassard and C. Crépeau. Zero-knowledge simulation of boolean circuits. In *Advances in Cryptology - proceedings of CRYPTO 86*, Lecture Notes in Computer Science, pages 223–233. Springer-Verlag, 1987.

[7] D. Chaum. Demonstrating that a public predicate can be satisfied without revealing any information about how. In *Advances in Cryptology - proceedings of CRYPTO 86*, Lecture Notes in Computer Science, pages 195–199. Springer-Verlag, 1987.

[8] 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.

[9] P.L. Chebyshev. Mémoire sur les nombres premiers. *J. Math. Pures et Appl*, (I)(17):366–390, 1852.

[10] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing*, pages 151–158, 1971.

[11] B. den Boer. An efficiency improvement to prove satisfiability with zero knowledge with public key. In *Advances in Cryptology - proceedings of EUROCRYPT 89*, Lecture Notes in Computer Science, 1989. To appear.

[12] U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, 1988.

[13] M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

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

[15] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.

[16] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. *SIAM Journal of Computation*, 18(1):186–208, 1989.

[17] R. Impagliazzo and M. Yung. Direct minimum-knowledge computations. In *Advances in Cryptology - proceedings of CRYPTO 87*, Lecture Notes in Computer Science, pages 40–51. Springer-Verlag, 1988.

[18] J. Kilian, S. Micali, and R. Ostrovsky. Efficient zero-knowledge proofs with bounded interaction. In *Advances in Cryptology - proceedings of CRYPTO 89*, Lecture Notes in Computer Science. Springer-Verlag, 1990. To appear.

[19] W. LeVeque. *Fundamentals of Number Theory*. Addison-Wesley, 1977.

[20] N. Pippenger and M. Fischer. Relations among complexity measures. *Journal of the Association for Computing Machinery*, 23:361–381, 1979.

[21] J. Rosser and L. Schoenfeld. Approximate formulas for some functions of prime numbers. *Illinois Journal of Mathematics*, 6:64–94, 1962.