An Introduction to Minimum Disclosure

G. Brassard
Département d'informatique et de R.O., Université de Montréal
C.P. 6128, Succursale "A", Montréal (Quebec), Canada H3C3J7

D. Chaum
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

C. Crépeau
Laboratory for Computer Science, Massachusetts Institute of Technology
545 Technology Square, Cambridge, MA 02139, U.S.A.

1. Introduction

Assume Peggy ('the Prover') knows some information. For instance, this could be the proof of a theorem or the prime factorization of a large integer. Assume further that Peggy's information is *verifiable,* in the sense that there exists an efficient procedure capable of certifying its validity. In order to convince Vic ('the Verifier') of this fact, Peggy could simply reveal the information to him so that he could perform the certifying procedure himself. This would be a *maximum disclosure* proof, since it results in Vic learning all the information. He could therefore later show it to someone else and even claim it to have been his originally.

In this paper we present a general and efficient protocol for obtaining *minimum disclosure* proofs. This protocol allows Peggy to convince Vic, beyond any reasonable doubt, that she has information that would pass the certifying procedure, but in a way that does not help him determine this information. For example, if Peggy's information is the proof of a theorem, Vic is left with the conviction that Peggy knows how to prove it, and hence that the theorem is true. However, Vic is not given even a clue as to how the proof might proceed (except perhaps for an upper limit on its length). Although Peggy's original information is verifiable, the conviction thus obtained by Vic may not be. In particular, conducting the protocol with Peggy need not (and in many cases *will* not) enable Vic to subsequently convince someone else.

At the heart of our protocol is the notion of *bit commitment,* which allows Peggy to commit herself to the value of some bits in a way that prevents Vic from learning them without her help. Bit commitment is implemented through our main primitive, which we call for convenience the *'blob'*. As this paper shows, the blob is a *universal* primitive for minimum disclosure. Each blob is used by Peggy as a commitment to either 0 or 1. For the sake of generality,

we do not impose any restriction on the nature of blobs - they could be made out of fairy dust if this were useful. By 'Peggy commits to a blob', we mean that Peggy has a blob 'in mind' and that she does something that will force her to stick to this blob in the future. If the blob itself can be represented as a bit string - as in most practical cases - committing to a blob can be as simple as showing it in the clear. The abstract defining properties of blobs are as follows:

i) Peggy can commit to blobs: by committing to a blob, she is in effect committing to a bit.
ii) Peggy can *open* any blob she has committed to: she can convince Vic of the value of the bit she in effect committed to when she committed to the blob. Thus, there is no blob she is able to 'open' both as 0 and as 1.
iii) Vic cannot learn anything about which way Peggy is able to open any unopened blob she has committed to. This remains true even after other blobs have been opened by Peggy.
iv) Blobs do not carry 'side information': the blobs themselves as well as the processes by which Peggy commits to and opens them are uncorrelated with any secret she wishes to keep from Vic.

Consider the following illustrative implementation of a blob. When Peggy wishes to commit to a bit (property (i)), she writes it on the floor and, before allowing Vic to look, she covers it with opaque tape. Although Vic cannot tell which bit is hidden under the tape (property (iii)), Peggy can no longer change it. To 'open the blob' (property (ii)), Peggy allows Vic to strip off the tape and look at the bit. Property (iv) is satisfied provided the way in which the bit is written on the floor, the tape and its placement are all uncorrelated with any secret Peggy wishes to keep from Vic.

In Section 2, we assume that blobs are available and show how to use them to obtain general minimum disclosure proofs. A discussion of this protocol follows in Section 3. In Section 4, we describe one specific implementation for blobs, which protects Peggy's information unconditionally but would allow her to lie to Vic by breaking some cryptographic assumption while the protocol is taking place. Once the protocol is over, it is too late for either party to attempt any kind of cheating, regardless of their computing power. Other possible implementations of blobs are given in [3], including dual implementations that are unconditionally secure for Vic, but could allow him to recover Peggy's information after some long (perhaps infeasible) off-line computation. Yet other implementations, also mentioned in [3], show neither weakness. Section 5 surveys the history of minimum disclosure. Finally, Section 6 compares the various implementations discussed in [3].

2. The protocol

Assume Peggy knows a satisfying assignment of truth values for some Boolean formula. Our protocol allows Peggy to convince Vic that she knows such an assignment without revealing any information about it. Because satisfiability of Boolean expressions is NP-complete [12,15], our protocol can be used to supply minimum disclosure proofs of knowledge for any positive statement

concerning a language L in NP. The protocol we describe here actually follows the lines of [9]. (Other constructions are given in [16,5], but [16] requires a reduction to a graph colouring problem and [5] requires that blobs satisfy additional properties. An extension to the case of *probabilistically* verifiable information is given in [3].)

As an example, consider the Boolean formula

$$\Psi = [(p \text{ and } q) \text{ xor } (\bar{q} \text{ or } r)] \text{ and } [(\bar{r} \text{ xor } q) \text{ or } (p \text{ and } \bar{r})]$$

and let $<p=true, q=false, r=true>$ be Peggy's secret satisfying assignment. (Of course, this is a toy example since it would be too easy for Vic or anyone else to find out how to satisfy such a simple Boolean expression.)

As a first step, Peggy and Vic agree on the layout of a Boolean circuit to compute Ψ. For simplicity, we use only basic binary gates and negations in the circuit. (Of course, negations are not needed, since any Boolean formula can be rewritten efficiently using only 'NAND' gates.) The circuit for Ψ is illustrated in Figure 1. In addition, this figure shows Peggy's satisfying assignment and the truth table of each gate (except the negation gates). Observe that one row is outlined in each truth table, corresponding to the circuit's computation on Peggy's satisfying assignment. Seeing the rows outlined is enough to easily verify that Ψ is satisfiable. This can be achieved by simple independent checks on the consistency of each wire. For instance, the output of the top left 'AND' gate is 0, which is indeed the first input of the middle row 'exclusive-or' gate. Also, the first input to the top left and top right 'AND' gates is the same, as it should be since they correspond to the same input variable. Finally, the output of the final gate is 1. Notice that seeing these outlined rows also gives away the corresponding satisfying assignment (even if it were not written explicitly). Our protocol allows Peggy to convince Vic that she knows how to so outline one row in each truth table - without revealing any information about which rows they are.

This is achieved by an interactive protocol consisting of several rounds. In each round, Peggy 'scrambles' the circuit's truth tables and commits to a corresponding collection of blobs. At this point, Vic issues one of two possible challenges to Peggy: one challenge requires Peggy to show that the blobs really encode a valid scrambling of the circuit's truth tables; the other challenge requires Peggy to open the rows that would be outlined, assuming it is a valid scrambling. The challenges are thus designed in such a way that Peggy could meet *both* of them only if she knew how to satisfy the circuit, but answering either *one* of them yields no information about how. Because Peggy cannot predict ahead of time which challenges will be issued by Vic, each round increases Vic's confidence in Peggy. In fact, Peggy would be caught cheating with probability at least 50% in each round if she were not able to answer both possible challenges, so that she could only hope to fool Vic in k successive rounds with exponentially vanishing probability 2^{-k}.

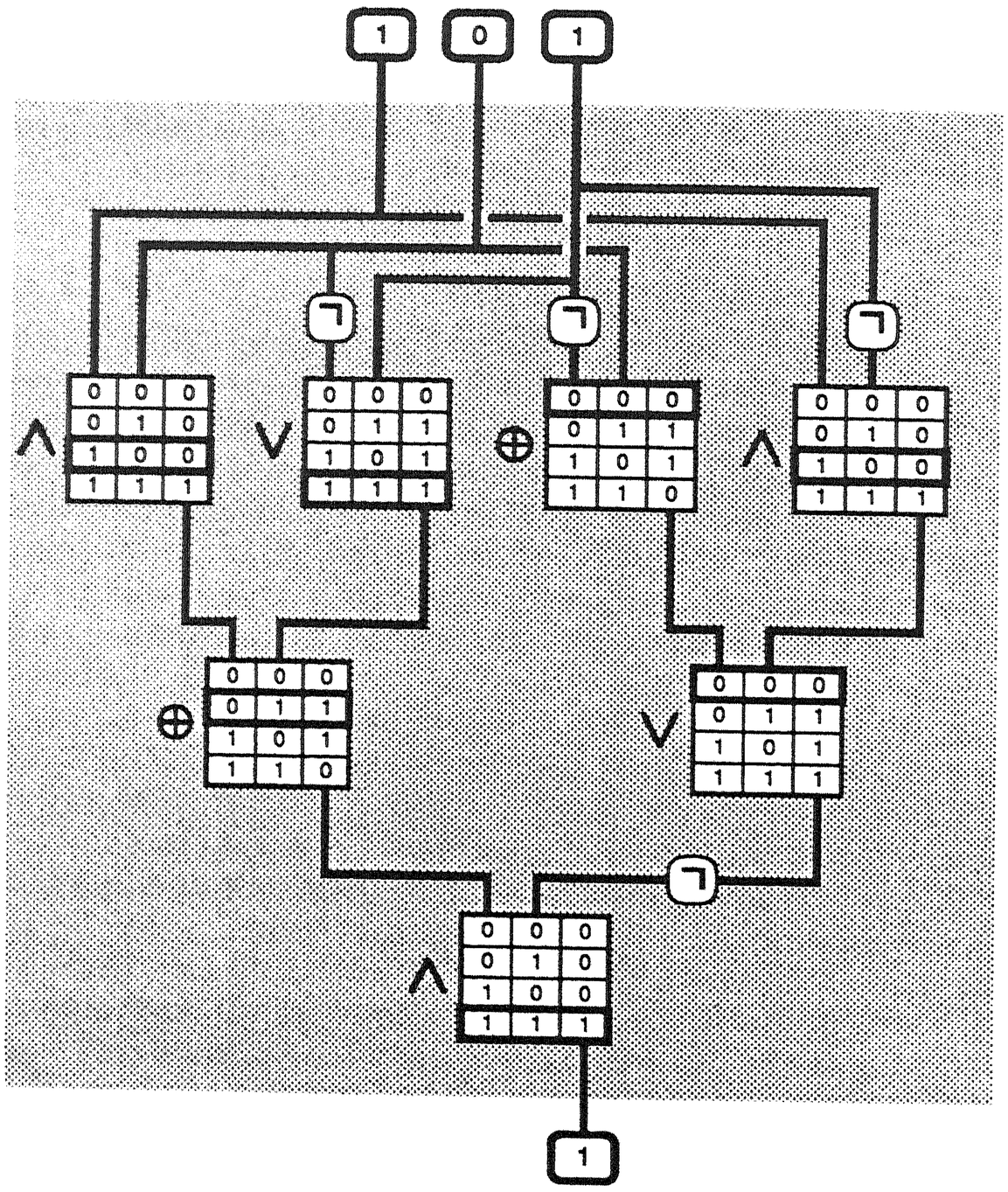

FIGURE 1. A Boolean circuit with explicit truth tables and rows outlined

We call such techniques 'cut-and-choose' because each round is similar to the classic 'protocol' by which two children split a piece of cake - one of them cuts and the other one chooses. The great utility of a cut-and-choose like ours is that it gives an exponential increase in security at the cost of only a linear increase in the number of rounds. The earliest use of such cut-and-choose we know of in the context of cryptographic protocols was presented by RABIN [19] in 1977.

The 'scrambling' of each truth table by Peggy consists of a random row permutation and column complementation. Let us illustrate this principle with an example. Figure 2(a) shows the truth table for the Boolean conjunction ('AND'). The rows of this table are randomly permuted to yield the table given in Figure 2(b). (Each of the 24 possible permutations - including the identity permutation - may be chosen with uniform probability.) Then, one bit is

randomly chosen for each of the three columns of the truth table. Finally, each column is complemented if and only if its corresponding random bit is a 1, as shown in the three intervening tables. The final result is illustrated in Figure 2(c). Notice that the whole scrambled truth table can still be unmistakably recognized as representing the Boolean conjunction (provided the complementation bits, shown within circles throughout the drawings, are specified).

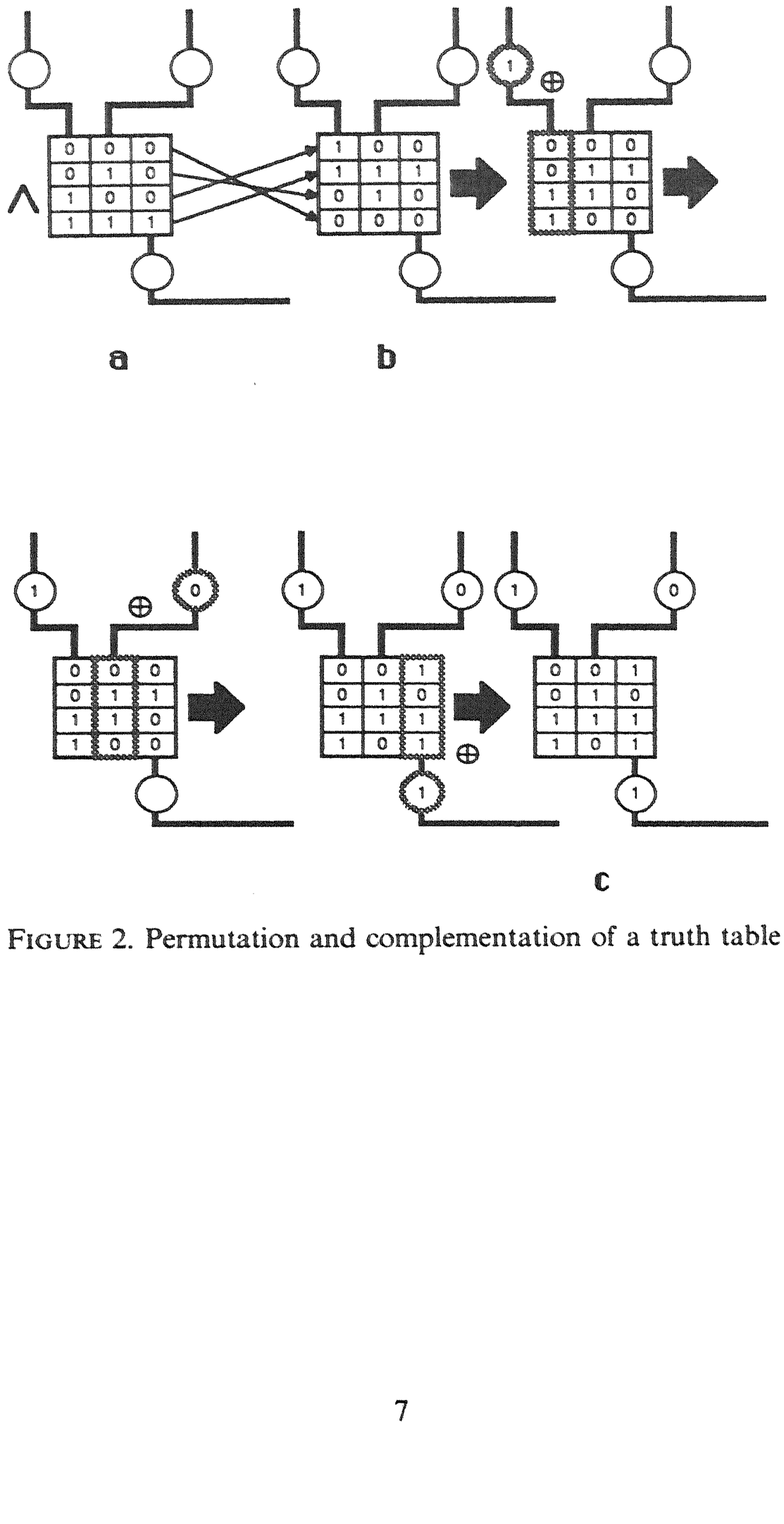

FIGURE 2. Permutation and complementation of a truth table

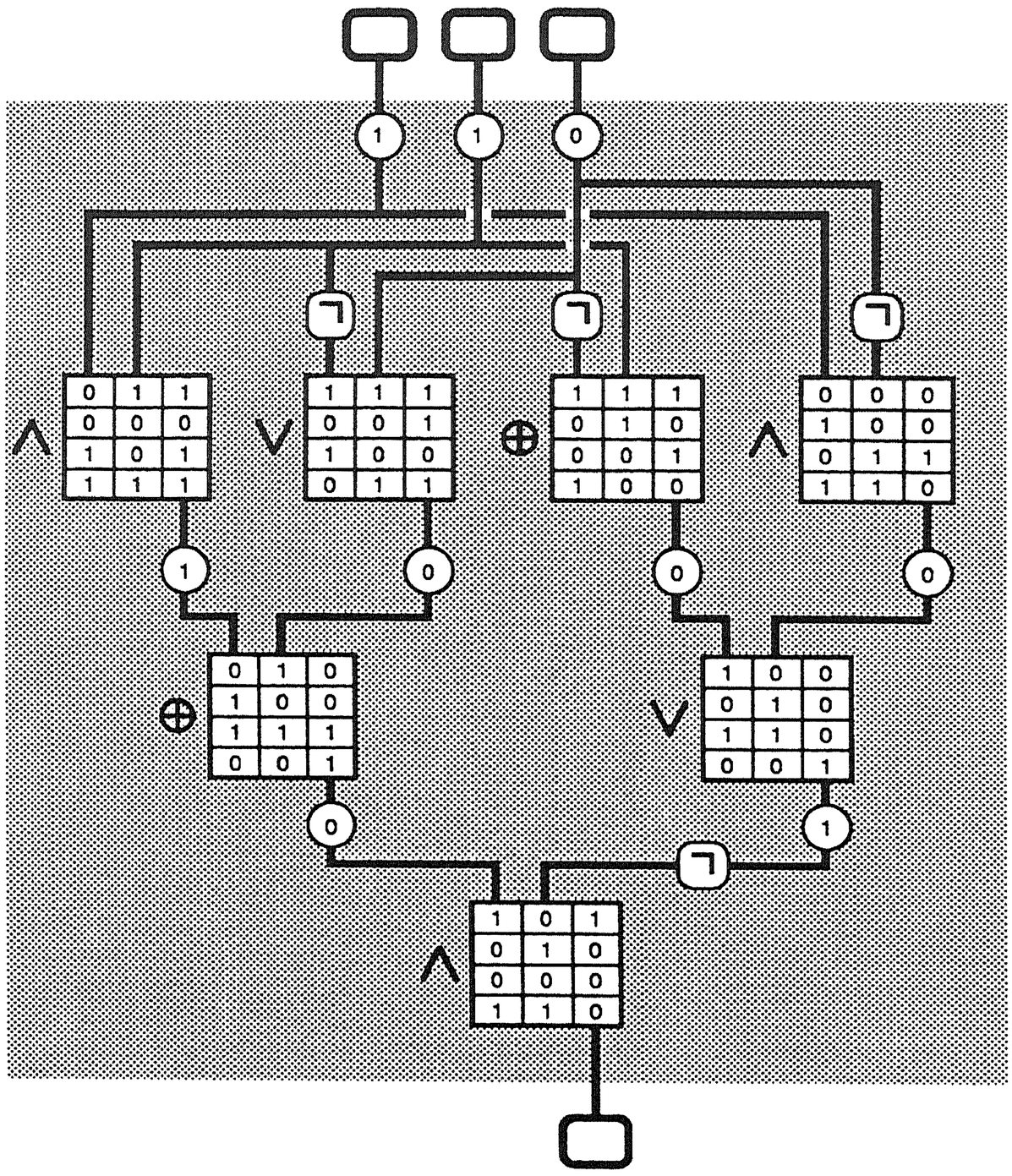

FIGURE 3. A circuit with randomly permuted and complemented truth tables

The complementations must be chosen consistently: all truth table columns corresponding to the same wire in the circuit must either all be complemented or all remain the same. This is achieved by choosing randomly and independently the complementation bits corresponding to each wire. (For simplicity, we never complement the output of the final gate.) Figure 3 gives the result of some example random permutations and complementations of the truth tables in our original circuit from Figure 1.

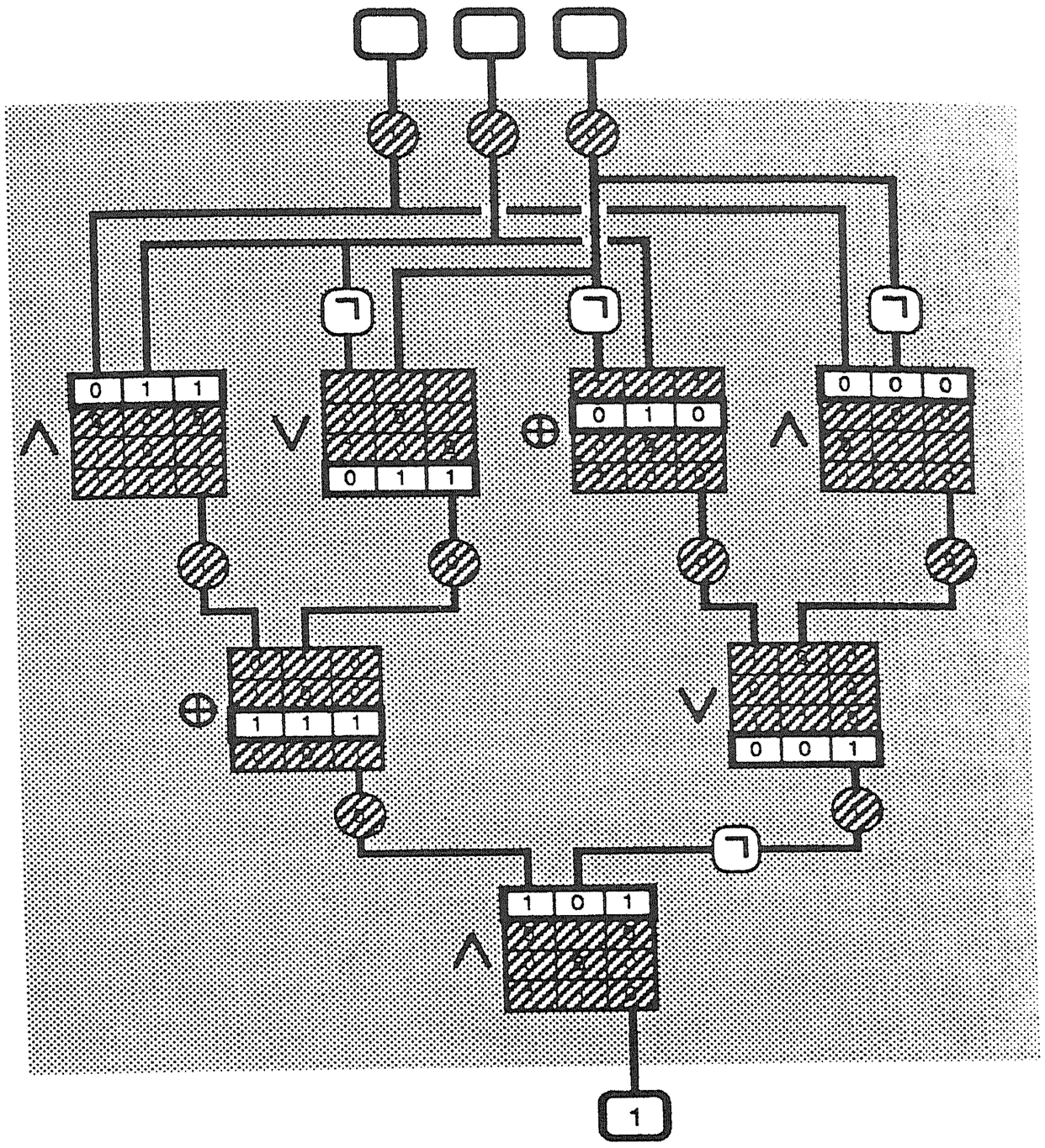

FIGURE 4. Showing the existence of a satisfying assignment

After producing a circuit similar to that of Figure 3, Peggy commits to it: for each truth table bit, Peggy commits to a blob that she knows how to open accordingly. (It is not necessary for her to actually commit to the complementation bits, but they must remain secret for the moment.) Coming back to our previous example blobs, one may think of Peggy having drawn Figure 3 on the floor but having covered its bits with opaque tape before allowing Vic to look. Now that the 'cut' is completed by Peggy, it is time for Vic to 'choose': Vic asks Peggy to convince him of her good faith by requesting that she answers, at his random choice, either Challenge 'A' or Challenge 'B', defined as follows:

- If the challenge is 'A', Peggy must open each and every blob she just committed to. Moreover, she must also reveal all the complementation bits that she used in the scrambling process. Continuing our intuitive image, Peggy strips off all the tape in order to show Vic the equivalent of Figure

3. This allows Vic to verify that the information concealed by the blobs corresponds to valid permutations and complementations of the Boolean circuit's truth tables.

- If the challenge is 'B', Peggy opens only the blobs corresponding to one row in each truth table. The rows to be opened are precisely those that were outlined in Figure 1 in their (probably) new location determined by the row permutations. Still continuing our image, Peggy selectively strips off pieces of tape in order to show Vic the equivalent of Figure 4. This allows Vic to verify the consistency of each wire and the fact that the final output of the circuit is a 1 bit.

3. Security of the protocol

Three requirements must be satisfied in order to prove the correctness of this protocol. Assuming only the four defining properties of blobs, it is proved in [3] that the following holds except perhaps with an exponentially small probability:

1) Peggy can carry out her share of the protocol, provided she knows a satisfying assignment for Ψ. (Of course, no protocol could possibly *force* Vic to be convinced, even giving him the satisfying assignment in the clear, because he can always refuse to listen.)
2) If Peggy does not know a satisfying assignment for Ψ, no matter how she pretends to follow the protocol, Vic will catch her cheating.
3) If Peggy knows a satisfying assignment for Ψ and if she faithfully follows her share of the protocol, she does not reveal anything to Vic that could help him determine her satisfying assignment (or even find partial information about it) - this remains true even if Vic deviates arbitrarily from his stipulated behaviour in the protocol.

Even though the third requirement is satisfied by our protocol, this does *not* in general imply that Vic cannot obtain anything beyond the fact that Peggy genuinely knows a satisfying assignment for Ψ. For instance, it is possible that only Peggy has the technology or knowledge necessary to commit to these blobs, in which case Vic might obtain something he could not have produced himself - although not the satisfying assignment. In other words, our protocol is minimum disclosure, but it may not be 'zero-knowledge' in the terminology of [18].

Intuitively, a protocol is *zero-knowledge* if the third requirement is strengthened to the effect that Vic cannot obtain anything at all beyond learning that Peggy knows a satisfying assignment. More precisely, Vic must be able to simulate his entire conversation with Peggy without in reality ever talking to her. Refer to [18] for a formal definition. Nevertheless, our protocol is zero-knowledge provided blob defining property (iv) is strengthened to make sure that Vic does not gain *anything* from the process by which Peggy commits to blobs and that he obtains *only* the intended bits from the process by which Peggy opens some of them. Following the proof techniques of [18], we say that the blobs are *simulatable* if, in addition to properties (i), (ii) and (iii), they satisfy

iv′) Vic can simulate what he would have been provided in the process by which Peggy commits to blobs that she could open as 0 and to blobs that she could open as 1. He can also simulate the process by which she would open these blobs had she committed to them herself.

A more interesting situation occurs if one considers a variation on the protocol in which all the rounds are carried out in parallel: Peggy commits all at once to blobs corresponding to k circuits similar to Figure 3, Vic sends his string of challenges, and Peggy opens the blobs as requested by the challenges. (This would be more efficient in some settings.) Assume for simplicity that blobs are bit strings and that Peggy commits to a blob by showing it in the clear. Consider the following strategy for Vic: after receiving from Peggy blobs corresponding to all k circuits with randomly permuted and complemented truth tables, he concatenates these blobs together and uses the result as input to some 'one-way' function. He then uses the first k bits of the output of this function to determine the k challenges to be issued. Even though running the modified protocol with Peggy does not help Vic in learning anything about Peggy's secret, its transcript may enable him to convince someone else of the existence of this secret, because Vic could almost certainly not have produced the transcript otherwise (even if the blobs are simulatable). This leads to a curious phenomenon: the transcript of a parallel version of the protocol may contain no information on Peggy's secret (in the sense of Shannon's information theory [20]), yet it can be used to convince someone else of the secret's existence!

If it is important that the protocol be carried out in parallel (perhaps for reasons of efficiency), it remains zero-knowledge provided defining property (iv) is strengthened further. We say that the blobs are *chameleon* if, in addition to properties (i), (ii) and (iii), they satisfy:

iv″) Vic can simulate what he would have seen in the process by which Peggy commits to blobs. Moreover, for each of these blobs, Vic can simulate both the process by which Peggy would open it as a 0 and the process by which she would open it as a 1.

In other words, chameleon blobs allow Vic to do just what property (ii) prevents Peggy from doing. Even if Peggy and Vic have similar computing abilities, this property can sometimes be achieved if Vic has additional information. The advantage of chameleon blobs is that they allow Vic to simulate in a straightforward way his entire conversation with Peggy. This remains true even if Vic deviates arbitrarily from his stipulated behaviour. Consult [3] for more details.

4. Blob implementations

We have taken the existence of blobs for granted in the previous sections. Let us now see how they can be implemented in practice. This can be done in several ways. None of these implementations is ideal, however. The choice of implementation should be based on the particular requirements of the application. Here, we present only one of the several implementations described in [3]. The safety of most of these implementations depends on unproved

assumptions about the computational difficulty of solving particular problems.

Some blobs are *unconditionally secure for Peggy*. In this case, defining property (iii) holds regardless of Vic's computing power. This is achieved by asking that it is not the blob itself that determines a bit, but rather Peggy's knowledge about it. Moreover, the probability that Peggy commits to any given blob must be the same whether she wishes to commit to 0 or to 1. This implies that Vic cannot learn anything about which way Peggy is able to open any unopened blob she has committed to. However, it also implies that Peggy could in principle violate property (ii), but the implementations are designed to make this computationally infeasible for her (under suitable assumptions).

Other blobs are *unconditionally secure for Vic*. In this case, property (ii) holds regardless of Peggy's computing power. This is achieved by asking that each blob can be opened to show only one possible bit. This implies that Peggy is irrevocably committed to a specific bit each time she commits to a blob. However, it also implies that Vic could in principle violate property (iii), but our implementations are designed to make this computationally infeasible for him (again under suitable assumptions).

Yet other blobs are secure even if all parties have unlimited computing power, but they rely on dogmas of quantum physics or require a multiparty environment under the assumption that the honest participants outnumber the cheaters.

Some elementary computational number theory is necessary to understand the implementation we have chosen to describe here. (This implementation is inspired by [10] and was independently proposed by [2].) Let p be a large prime and let α generate $\mathbb{Z}_p^*$, the multiplicative group of integers modulo p. Given any integer y, it is easy to compute $\alpha^y \bmod p$, but no efficient algorithm is known to invert this process, an operation known as computing the 'discrete logarithm modulo p'. The intractability assumption of the discrete logarithm was used in the very first paper published on public-key cryptography [13]. It can be used also to create blobs provided it is strengthened to assume that computing discrete logarithms modulo a large prime p remains infeasible even if the factorization of $p-1$ is known.

At the outset of the protocol, Peggy and Vic agree on a suitable prime number p for which both of them know the factorization of $p-1$. They also agree on α, a generator of the group $\mathbb{Z}_p^*$. Thanks to their knowledge of the factors of $p-1$, they can both verify with certainty that p is a prime and that α is a generator of $\mathbb{Z}_p^*$. These same parameters p and α can be public, in the sense that they can be used with no breach of security by all parties wishing to engage in minimum disclosure protocols. At the outset, Vic also chooses a random $s \in \mathbb{Z}_p^*$ $(s \neq 1)$ and gives it to Peggy. Assuming the intactability of the discrete logarithm, Peggy cannot compute e such that $s \equiv \alpha^e \pmod p$.

In order to commit to some bit b, Peggy chooses a $y \in Y$ randomly and computes $x = s^b \alpha^y \bmod p$. She gives x - the blob - to Vic but keeps y secret as her *witness* that allows her to open x as bit b. Clearly, any element of $\mathbb{Z}_p^*$ can be used by Peggy as commitment to 0 just as well as to 1, depending only on her knowledge about it. Therefore, property (iii) holds unconditionally: blobs

committed to by Peggy contain no information on the way in which she could open them. Property (ii) holds computationally because Peggy could easily obtain e (which we assumed to be infeasible for her) from knowledge of y_0 and y_1 such that $\alpha^{y_1} = s\alpha^{y_2} \pmod{p}$. Moreover, these blobs are obviously simulatable.

Using the terminology of [16], this implementation of blobs turns the protocol presented here into a 'perfect zero-knowledge interactive protocol' for satisfiability (except that it does not fit their model as an interactive protocol since they allow the Prover to be infinitely powerful, in which case she would have no problem computing e - which explains why Fortnow's theorem [14] does not apply). Such a perfect zero-knowledge interactive protocol was *incorrectly* claimed in [5] about another implementation of blobs.

The 'discrete logarithm blobs' are not chameleon, and thus our protocol should not directly be performed in parallel if it is to be zero-knowledge. If it were performed in parallel, Vic could cheat by choosing a random integer e and computing $2\alpha^e \bmod p$ as the s he gives to Peggy. Assume Peggy uses the parallel version of the protocol to convince Vic that she knows the proof of some theorem T. If Vic uses a one-way function for example to select his challenges, he could subsequently use the transcript, together with the value of e, to convince others that T is true. Indeed, there is no obvious way by which Vic could have created this transcript by himself, unless he knows a proof of T or the discrete logarithm of 2. Again, this illustrates a very curious phenomenon: although the transcript of the protocol can be used as evidence that T is true, it cannot be used in any way to facilitate finding such a proof. Moreover, the transcript contains no information on the proof of T, even in the sense of Shannon's information theory [20].

With some preprocessing, it is possible to add the chameleon property to these blobs. Rather than choosing s randomly in $\mathbb{Z}_p^*$, Vic chooses an integer e randomly between 1 and $p-2$ and computes $s = \alpha^e \bmod p$. Using a minimum disclosure protocol [11], he then convinces Peggy that he knows the discrete logarithm of s, which is all he needs to meet property (iv''). Note that in this case Vic would also convince Peggy that s is in the subgroup generated by α, so that the requirement that α be a generator of $\mathbb{Z}_p^*$ is no longer crucial for Peggy's safety. Therefore, if we tolerate an exponentially small probability that Vic could gain information on Peggy's secret, the factorization of $p-1$ need not be known and thus the assumption about the difficulty of computing discrete logarithms can be relaxed.

5. Related work

As occurs often in research, some of the ideas presented here were developed independently in several places. An early interactive proof was presented by Rabin [19], as already mentioned. This concept was formalized and the notion of 'zero-knowledge' protocols (which is related to minimum disclosure) was introduced in [18]. Also, [1] formalized a notion similar to that of interactive proofs. The model proposed in [18,1] is quite interesting from a theoretical point of view, but it is based on the assumption that the prover has

unlimited computing power.

Assuming only the existence of secure probabilistic encryption schemes (in the sense of [17]), [16] showed that 'Every language in NP has a zero-knowledge interactive proof system in which the prover is a probabilistic polynomial-time machine that gets an NP proof as an auxiliary input'. Under a stronger assumption, the same result was obtained independently but subsequently in [4]. A similar result was also obtained independently by [9], but in a very different model, which emphasizes the unconditional privacy of the prover's secret information, even if the verifier has unlimited computing resources. This model was set forward in [7] and the result of [9] is a special case of a protocol previously presented in [6], whose properties are described in [7, page 1039]. (The results of [9] (then [8]) and [16] were first presented explicitly in March 1986 at the Marseille conference on Algorithms, Randomness and Complexity.) Finally, [5] considered a model in which all parties involved are assumed to have 'reasonable' computing power (this model is also compatible with the setting of [9]). All these approaches are unified in [3].

The difference between these models can be illustrated by an example. Consider the statement by which Peggy claims to know the prime factorization of some public integer n. In the [18] model, there would be no point in her spending time convincing Vic of this, because Vic knows that it is an immediate consequence of her unlimited computing power. In the setting of [9], her secret factorization cannot possibly be unconditionally secure once the integer n is made public; she may therefore just as well convince Vic that she knows the factors by giving them explicitly to Vic. (But if Peggy's statement had merely been that she knows non-trivial divisors of n, and if n is the product of several primes, the setting of [9] would allow Peggy to convince Vic of her knowledge without disclosing any information as to which divisors she knows, even if Vic has unlimited computing power.) In the context of [5], on the other hand, it makes perfect sense for Peggy to wish to convince Vic of her knowledge via a protocol that does not disclose anything that could help Vic compute the factors of n. In other words, the protocol is designed to make Vic's factoring task just as difficult after the protocol as it was before.

6. Is it better to trust the prover or the verifier?

'Cheating' takes on a different meaning, depending on whether one is talking about Peggy or Vic. For Vic to cheat means that he learns something beyond the fact that Peggy has access to the information she claims to have. Perhaps he did not quite obtain the Hamiltonian circuit he is desperately seeking, for instance, but he learned enough to drastically reduce his search. On the other hand, for Peggy to cheat means that she succeeds in convincing Vic that she has information that would pass the certifying procedure, when in fact she does not.

It is also interesting to distinguish between *lucky* and *daring* successful cheating. The former refers to Peggy or Vic figuring out - against all odds - a piece of information that will enable him/her to quietly go about his/her cheating with the certainty of being successful and undetected. The latter

refers to Peggy or Vic taking an illegal move that is almost certainly going to result in his/her cheating being detected at some point in the future, but that might nonetheless, with an exponentially small probability, allow him/her to succeed. Finally, cheating may be called *retroactive* (or *off-line*) if it can take place some time after the protocol is completed, by looking back at its transcript; it is *real-time* if it must be completed while the protocol is taking place.

If blobs unconditionally secure for Vic are used (corresponding to the protocols previously given by [16,4]), Peggy could never participate with peace of mind: an algorithmic breakthrough might allow Vic to cheat retroactively, even if the new algorithm is not fast enough for a real-time response while the protocol is taking place. Even if the cryptographic assumption turns out to be well-founded, Vic still has a (very slight) probability of lucky (hence undetectable) cheating. On the other hand, regardless of any assumptions, the only cheating Peggy could attempt would be of the daring kind.

By contrast, if blobs unconditionally secure for Peggy are used (corresponding to the protocols previously given in [9,5]), Vic's belief that Peggy cannot cheat depends on his belief in the appropriate cryptographic assumption. With the implementation of blobs described in the previous section, for instance, Peggy could 'open' any blob as either 0 or 1, whichever suits her best, if she could only obtain the discrete logarithm of s before the end of the first round in which she is asked a challenge she is not otherwise prepared to answer. (Obtaining this logarithm at any later time would be of no use to her.) Moreover, even if the cryptographic assumption is well-founded, Peggy still has a (very slight) possibility of breaking it by luck, but she must be daring to suggest conducting the protocol in the hope that she will be so lucky. With the discrete logarithm blobs, Vic has no possibility of cheating whatsoever, regardless of his luck and computing power. Finally, retroactive cheating is meaningless for either party in this context.

If blobs unconditionally secure for Peggy are used, additional security for Vic is obtained by asking Peggy to repeat the entire protocol with a different type of blob each time (as pointed out originally in [9]). In order to cheat, this would force Peggy to be capable of breaking several different cryptographic assumptions. For instance, using also the blob implementation of [5], she would need efficient on-line algorithms both for factoring and for extracting discrete logarithms. Curiously, the opposite effect is obtained with the blobs that are unconditionally secure for Vic: repeating the protocol with different types of blobs would only make it *easier* for Vic to cheat since he can do so by breaking (possibly off-line) any one of the underlying cryptographic assumptions. Nonetheless, increased security *can* be obtained if several types of blobs unconditionally secure for Vic are combined in a different way: each time Peggy wishes to commit to some bit b, she commits to one blob of each type at random except that b is the exclusive-or of the corresponding bits. Naturally, using this strategy with blobs unconditionally secure for Peggy would only make it easier for her to cheat.

Is it preferable to trust Vic or Peggy? We do not know, but it sure is nice to have the choice! Finally, consider the following provocative situation: suppose

that Peggy claims to have proven Theorem T and she uses the discrete logarithm blobs to convince a skeptical Vic of this. At the end of the protocol, regardless of any unproved assumptions, Vic will be convinced that Peggy has either a proof of T or hot results on extracting discrete logarithms! In particular, no assumptions are needed if T's claim is: 'I have an efficient algorithm for the discrete logarithm problem'...

ACKNOWLEDGEMENT

We wish to thank Josh Benaloh, Charles H. Bennett, Joan Boyar, Ivan Damgård, Jan-Hendrik Evertse, Joan Feigenbaum, Lance Fortnow, Shafi Goldwasser, Oded Goldreich, Jeroen van de Graaf, Johan Hastad, Russel Impagliazzo, Leonid Levin, Silvio Micali, Bill Neven, Jean-Marc Robert, Steven Rudich, Adi Shamir and Moti Yung for fruitful discussions.

REFERENCES

1. L. BABAI (1985). Trading group theory for randomness. *Proceedings of the 17th Annual ACM Symposium on the Theory of Computing,* 421-429.
2. J.F. BOYAR, M.W. KRENTEL, S.A. KURTZ (1987). *A Discrete Logarithm Implementation of Zero-Knowledge Blobs,* Technical report 87-002, University of Chicago.
3. G. BRASSARD, D. CHAUM, C. CRÉPEAU (1987). *Minimum Disclosure Proofs of Knowledge,* Technical Report PM-R8710, Centre for Mathematics and Computer Science (CWI), Amsterdam.
4. G. BRASSARD, C. CRÉPEAU (1987). Zero-knowledge simulation of Boolean circuits. *Advances in Cryptology*: *Proceedings of* CRYPTO 86, Santa Barbara, California, 223-233.
5. G. BRASSARD, C. CRÉPEAU (1986). Non-transitive transfer of confidence: a *perfect* zero-knowledge interactive protocol for SAT and beyond. *Proceedings of the 27th Annual IEEE Symposium on the Foundations of Computer Science,* 188-195.
6. D. CHAUM (1985). *Showing Credentials without Identification: Signatures Transferred between Unconditionally Unlinkable Pseudonyms.* Presented at EUROCRYPT 85, Linz.
7. D. CHAUM (1985). Security without identification: transaction systems to make Big Brother obsolete. *Communications of the ACM 28,* 1030-1044.
8. D. CHAUM (1986). *Showing Satisfiability without Revealing How.* Presented at the Conference on Algorithms, Randomness and Complexity, Marseille/Lumigny.
9. D. CHAUM (1987). Demonstrating that a public predicate can be satisfied without revealing any information about how. *Advances in Cryptology: Proceedings of* CRYPTO '86, Santa Barbara, California, Springer-Verlag, 195-199.
10. D. CHAUM, I.B. DAMGÅRD, J. VAN DE GRAAF Multiparty computations ensuring privacy of each party's input and correctness of the result. *Advances in Cryptology: Proceedings of* CRYPTO '87, Santa Barbara, California, Springer-Verlag. To appear.

11. D. CHAUM, J.-H. EVERTSE, J. VAN DE GRAAF, R. PERALTA (1987). Demonstrating possession of a discrete logarithm without revealing it. *Advances in Cryptology: Proceedings of* CRYPTO '86, Santa Barbara, California, Springer-Verlag, 200-212.
12. S.A. COOK (1971). The complexity of theorem proving procedures. *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing,* 151-158.
13. W. DIFFIE, M.E. HELLMAN (1976). New directions in cryptography. *IEEE Transactions on Information Theory IT-22,* 644-654.
14. L. FORTNOW (1987). The complexity of perfect zero-knowledge. *Proceedings of the 19th Annual ACM Symposium on the Theory of Computing,* 204-209.
15. M.R. GAREY, D.S. JOHNSON (1979). *Computers and Intractability: a Guide to the Theory of NP-Completeness,* W. H. Freeman and Company, New York.
16. O. GOLDREICH, S. MICALI, A. WIGDERSON (1986). Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. *Proceedings of the 27th Annual IEEE Symposium on the Foundations of Computer Science,* 174-187; originally presented at the Conference on Algorithms, Randomness and Complexity, Marseille/Lumigny.
17. S. GOLDWASSER, S. MICALI (1984). Probabilistic encryption. *Journal of Computer and System Sciences 28,* 270-299.
18. S. GOLDWASSER, S. MICALI, C. RACKOFF (1985). The knowledge complexity of interactive proof-systems. *Proceedings of the 17th Annual ACM Symposium on the Theory of Computing,* 291-304.
19. M.O. RABIN (1978). Digitalized signatures. R.A. DEMILLO, D.P. DOBKIN, A.K. JONES, R.J. LIPTON (eds.). *Foundations of Secure Computation,* Academic Press, New York, 155-168.
20. C.E. SHANNON (1948). A mathematical theory of communication. *Bell System Technical Journal,* 379-423, 623-656.