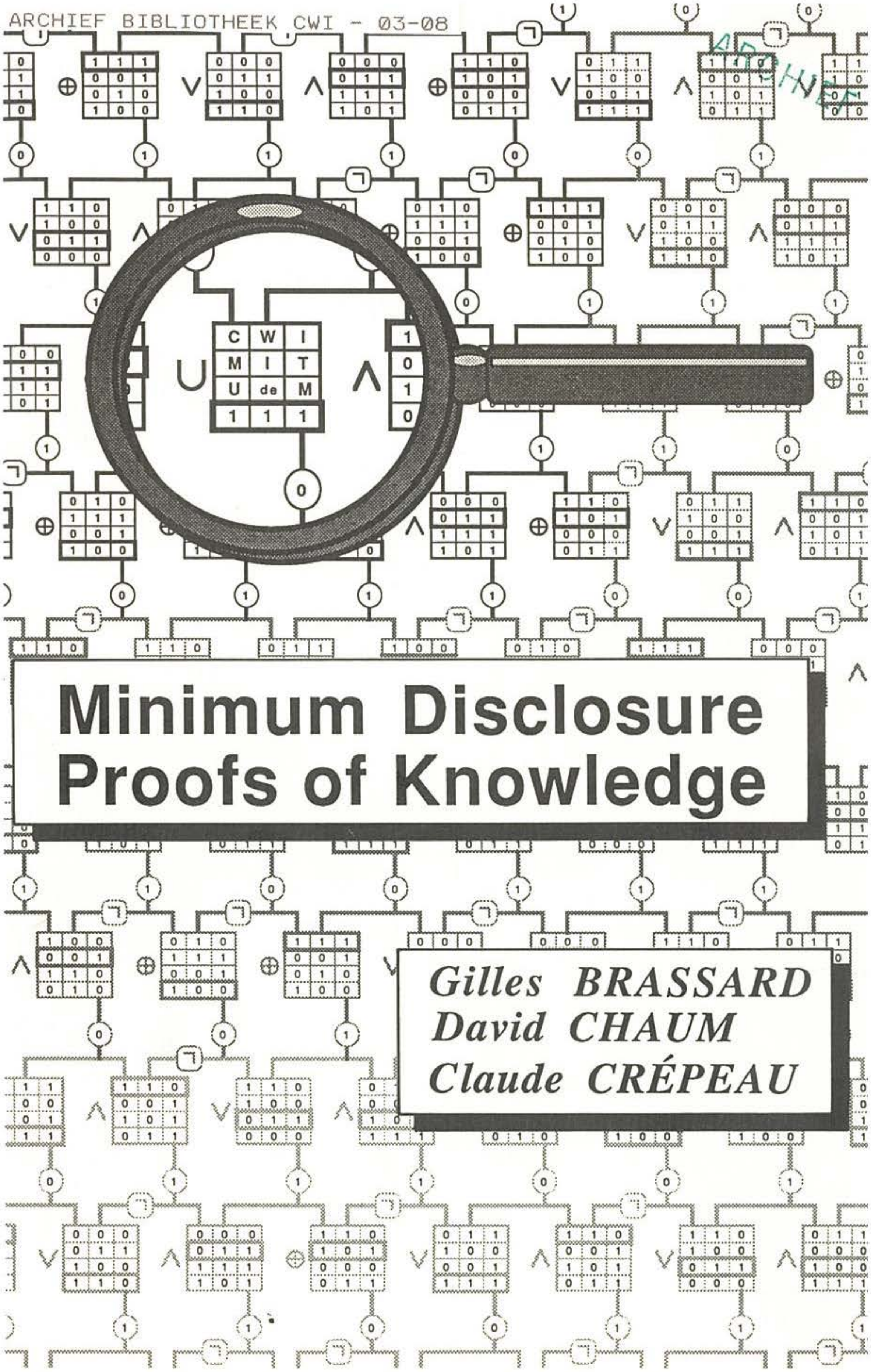

Minimum Disclosure Proofs of Knowledge

Gilles BRASSARD
David CHAUM
Claude CRÉPEAU

Minimum Disclosure Proofs of Knowledge

Gilles BRASSARD[†]
David CHAUM[*]
Claude CRÉPEAU[‡]

July 1987

[†] Département d'informatique et de R.O.
Université de Montréal
C.P. 6128, Succursale "A"
Montréal (Québec)
Canada H3C 3J7

[*] Centre for Mathematics and Computer Science (CWI)
Kruislaan 413
1098 SJ Amsterdam
the Netherlands

[‡] Laboratory for Computer Science
Massachusetts Institute of Technology
545 Technology Square
Cambridge, MA 02139
U.S.A.

[†] Supported in part by NSERC grant A4107.
Research conducted in part at CWI.

[‡] Supported in part by an NSERC postgraduate scholarship.
Supported in part by NSF grant DCR MCS8509905.
Research conducted in part at Université de Montréal and CWI.

1. Introduction

Assume Peggy ("the Prover") knows some information. For instance, this could be the proof of some theorem or the prime factorization of some 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 another party Vic ("the Verifier") of this fact, Peggy could simply show him the information and let him perform the certifying procedure. This would be a *maximum disclosure* proof, since it results in Vic learning all the information. He could therefore subsequently show it to someone else and even claim it to have been his originally.

In this paper, we present very general *minimum disclosure* proofs. These allow Peggy to convince Vic, beyond any reasonable doubt, that she has information that would pass the certifying procedure, but in a way that discloses absolutely nothing more. For example, if Peggy's information is the proof of a theorem, Vic is left with the conviction that the theorem is true and that Peggy knows how to prove it, but not even a hint 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 enable Vic to subsequently convince someone else.

The notion of minimum disclosure proofs extends to the case of *probabilistically verifiable information*. Assume for instance that Peggy generates two distinct integers that pass a probabilistic primality test to her satisfaction. She computes their product, makes it public, and then claims that she knows its prime factorization. Does she have verifiable information to support her claim, considering the fact that she does not have a definite proof that her factors are primes? In a case like this, even though the efficient certifying procedure for her information is probabilistic, it still makes sense for her to use a minimum disclosure proof to convince Vic that her claim is true.

In this paper, we give a very general protocol for obtaining minimum disclosure proofs and several practical ways to actually implement it. This protocol applies as well to the case of probabilistically verifiable information. At the heart of our protocol is the notion of *bit commitment*, which allows Peggy to commit herself to the Boolean 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*". The defining properties of blobs are as follows:

i) Peggy can obtain blobs representing 0 and blobs representing 1.

ii) When presented with a blob, Vic cannot tell which bit it represents.

iii) Peggy can *open* blobs by showing Vic the single bit each represents; there is no blob she is able to "open" both as 0 and as 1.

Consider the following illustrative implementation of a blob. When Peggy wishes to commit to a bit, she writes it down on a piece of paper, slips it in an envelope, seals the envelope, and gives it to Vic. Upon receiving it, Vic writes an identifying serial number "*s*" on the envelope, applies his signature, and returns the envelope to Peggy. At this point, Peggy has succeeded in obtaining a blob (property (i)). Although Vic cannot tell which bit is hidden in envelope number "*s*" (property (ii)), Peggy can no longer change it (property (iii)). To "open the blob" (property (iii)), Peggy gives the envelope back to Vic, who checks his signature and the serial number before physically opening the envelope to look inside.

In the following sections, we shall assume that blobs are available and show how to use them to obtain general minimum disclosure protocols. Sections 2 and 3 deal with the case of deterministically verifiable information. After a complexity theoretic interlude in section 4, section 5 gives the general protocol for probabilistically verifiable information. Under various assumptions, we suggest in section 6 several implementations for blobs and compare their relative strengths and weaknesses. As we shall see, some blob implementations lead to protocols that protect Peggy's information unconditionally but that would allow her to lie to Vic by breaking some cryptographic assumption in real time. Dual blob implementations are unconditionally secure for Vic, but could allow him to recover Peggy's information after some long (perhaps infeasible) off-line computation. Other implementations show neither weakness, but rely on dogmas of quantum physics or require the participation of several parties.

1.1. Related work

As is quite common in research, some of the ideas presented here were developed independently in several places. The concepts of interactive proofs and "zero-knowledge" protocols were introduced in [GMR], a seminal paper that gives formal definitions for these notions. Also, [Ba] presented a notion similar to that of interactive proofs. The model they propose is very 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 [GM]), [GMW] showed that "Every language in **NP** has a zero-knowledge proof system in which the prover is a probabilistic polynomial-time machine that get an **NP** proof as auxiliary input". Under a stronger assumption, the same result was obtained independently but subsequently in [BC1]. A similar result was also obtained independently by [Ch3], but under 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 [Ch2] and the result of [Ch3] is a special case of a protocol previously presented in [Ch1], whose properties are described in [Ch2, page 1039]. Finally, [BC2] considered a model in which all parties involved are assumed to have "reasonable" computing power (this model is also compatible with the setting of [Ch3]). The current paper unifies all of these approaches.

The difference between these models can be illustrated by an example. Consider again the statement by which Peggy claims to know the prime factorization of some public integer n. In the [GMR] model, there would be no point for her to spend time convincing Vic of this because Vic knows that it is an immediate consequence of her unlimited computing power. In the setting of [Ch3], 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 a non-trivial divisor of n, and if n is the product of several primes, the setting of [Ch3] would allow Peggy to convince Vic of her knowledge without disclosing any information as to which proper divisors she knows, even if Vic has unlimited computing power.) In the context of [BC2], 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 in such a way as to make Vic's factoring task exactly as difficult after the protocol as it was before.

As we shall see in section 7, it is also interesting to distinguish between the parties' available computing resources *during* and *after* the protocol. Our main result is a protocol that is unconditionally secure for *both* parties as long as Peggy is incapable of factoring a large integer (or extracting a discrete logarithm, or both simultaneously) *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. This is in sharp contrast with the result of [GMW, BC1] concerning **NP**-complete problems, which allows Vic to take as much time as he likes to decipher the protocol transcript off-line and extract Peggy's secret.

2. The Basic Protocol

Let Ψ be a Boolean formula. Assume Peggy knows a satisfying assignment of truth values to its Boolean variables. The basic protocol allows Peggy to convince Vic that Ψ can be satisfied without revealing any information about how. This protocol follows the lines of [Ch3]. (Other constructions are given in [GMW, BC2], but [GMW] requires a reduction to a graph colouring problem and [BC2] requires that blobs satisfy additional properties.)

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). As a toy example, consider the Boolean formula

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

and let $< p = \textit{true},\ q = \textit{false},\ r = \textit{true} >$ be Peggy's secret satisfying assignment for it. 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

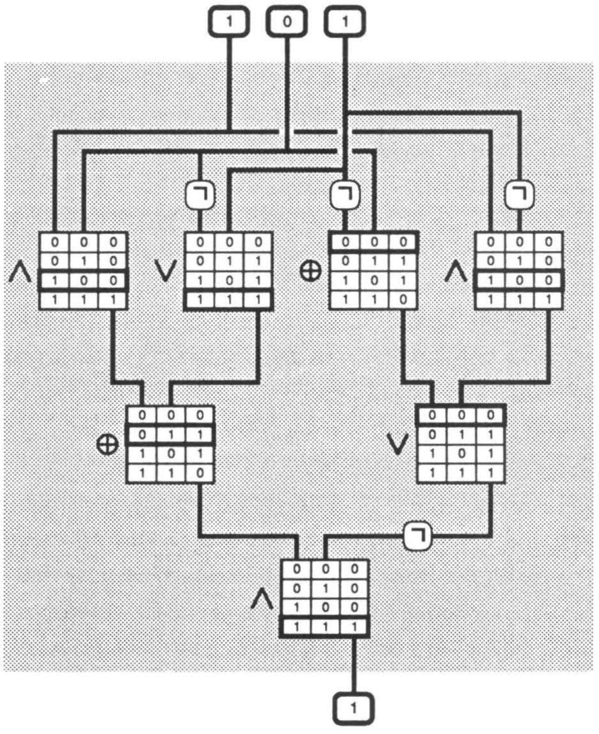

Figure 1. A Boolean circuit with explicit truth tables and rows outlined.

(apart from negations). 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 verify that Ψ is satisfiable. This is 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 rows outlined also gives away the corresponding satisfying assignment (even if it were not written explicitly). The basic protocol allows Peggy to convince Vic that she knows how to so outline one row in each truth table, but without revealing any information about which rows are concerned.

This is achieved by an interactive protocol that consists of several rounds. In each round, Peggy scrambles the circuit's truth tables and gives them to Vic as a collection of blobs. At this point, Vic issues one of two

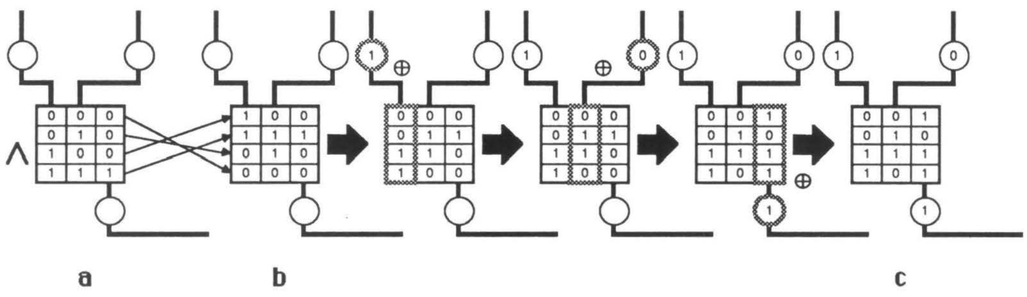

Figure 2. Permutation and complementation of a truth table.

possible challenges to Peggy: one challenge requires Peggy to show that the blobs really encode a valid scrambling of the circuit; 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. As a result, 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}.

The basic scrambling step for Peggy consists of a random row permutation and column complementation of the circuit's truth tables. 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 may be chosen with uniform probability — including the identity permutation.) 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 interveaning tables. The final result is illustrated in figure 2(c). Notice that the whole scrambled table, including the complementation bits (always shown within circles), can still unmistakably be recognized as representing the Boolean conjunction.

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 a random permutation and complementation of

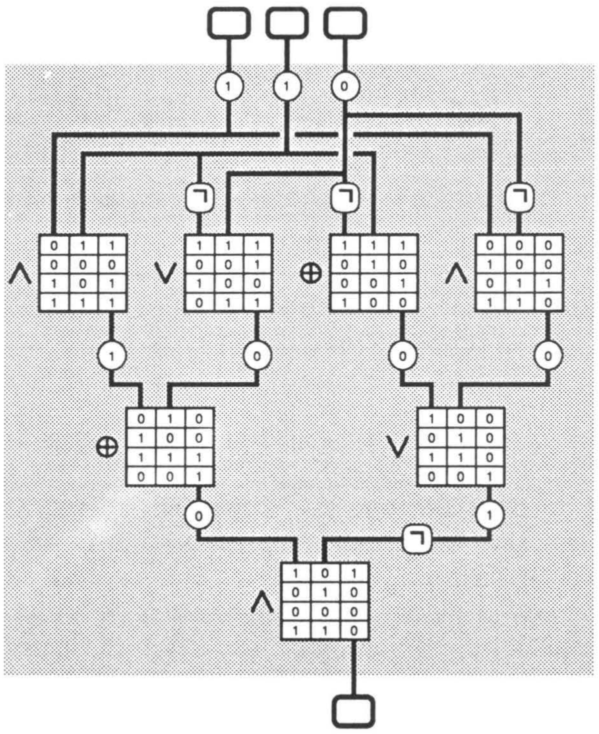

Figure 3. A randomly permuted and complemented circuit.

our original circuit from figure 1.

After producing this scrambled circuit, Peggy "shows" it to Vic, except that the bits are encoded as blobs: each truth table bit and each complementation bit is encoded as a blob. Intuitively, 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. At this point, Vic asks Peggy to convince him of her good faith by sending her at random either Challenge "A" or Challenge "B", defined as follows:

- If the challenge is "A", Peggy must open each and every blob she just gave Vic. 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 a valid permutation and complementation of the Boolean circuit.
- If the challenge is "B", Peggy opens only the blobs corresponding to one row in each truth table (taking account of the truth table row permutations, the rows to be opened are precisely those that were outlined in

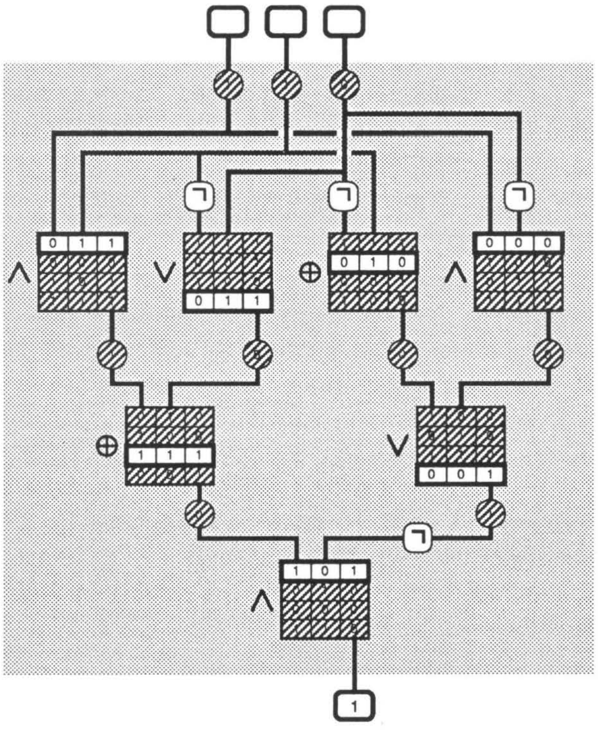

Figure 4. Showing the existence of a satisfying assignment.

figure 1). 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. Proof of the Basic Protocol

Three requirements must be satisfied in order to prove correctness of the basic protocol; the following must hold except perhaps with an arbitrarily small probability:

1) if Peggy knows a satisfying assignment for Ψ, then she can carry out her share of the protocol (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, she will be caught cheating by Vic; and

3) if Peggy knows a satisfying assignment for Ψ and if she follows faithfully her share of the protocol, nothing Vic can do will enable him to learn anything beyond the fact that Peggy genuinely knows a satisfying assignment.

No assumptions about the blobs are needed to see why the first requirement is met. This is simply due to the fact that knowing a satisfying assignment for Ψ allows Peggy to outline one row in each truth table. She can do this simply by remembering where the random truth table permutations have taken the rows that she must know would be outlined in the originally agreed circuit.

The second requirement is satisfied because of the bit commitment property (iii) of the blobs. Assume Peggy does not know how to satisfy Ψ. In any given round, she can either commit to a scrambled circuit (similar to figure 3) which is a genuine permutation and complementation of the original circuit, or she can commit to something phoney. In the first case, she cannot meet Challenge "B" without knowing a satisfying assignment for Ψ; in the second case, she cannot meet Challenge "A" without breaking the bit commitment property of blobs. Therefore, as long as she cannot predict the challenges to be issued by Vic, she has at least a 50% chance of being caught in each round. As mentioned earlier, her probability of fooling Vic in k successive rounds is therefore at best 2^{-k}.

The reason why the third requirement is satisfied is more subtle. Let us first argue that Vic cannot learn anything (beyond Peggy's honesty) from seeing either figure 3 or figure 4 alone. If he issues Challenge "A" and thus gets to see figure 3, he obtains a randomly permuted and complemented version of the Boolean circuit. This is of no possible use to Vic because he could have produced such a figure just as well by himself (even if the Boolean formula were *not* satisfiable). On the other hand, if he issues Challenge "B" in order to see figure 4, what he gets amounts to the result of applying a true one-time pad (Peggy's independent random complementations) on the Boolean values carried by the circuit wires while it computes a satisfying assignment (except for the final output wire, which should carry the value 1). This conveys no information either. In other words, it is only by matching a figure 4 with its corresponding figure 3 that Vic could hope to learn something, but of course Peggy will never answer both challenges in the same round.

At this point, the third requirement is proven for the floor-and-opaque-tape "implementation" used as intuitive image in the previous section. If we wish to consider blobs in general, however, a technical difficulty remains. Although Vic learns nothing from seeing any number of non-matching figure 3's and figure 4's, he might learn something from the blobs themselves. As it turns out, the three properties of blobs given in the introduction are *not* sufficient to prove that this cannot hapen in general. Indeed, they do not rule out the possibility for blobs to carry "side information". For instance, in addition to secretly encoding a bit as it should, nothing prevents a blob from giving in the clear Peggy's newly discovered proof of Fermat's "last" theorem! An additional constraint must therefore be imposed as a defining property for blobs. (This was disregarded in the introduction to avoid confusing the reader with a technicality that is not essential to the intuitive idea behind the basic protocol. The following requirement could be weakened if "minimum disclosure" only meant that running the protocol should not help Vic determine Peggy's secret; it is however necessary if, in addition, we require that Vic should obtain nothing that he could not have obtained by himself.)

iv) It is also possible for Vic to obtain blobs representing 0 and blobs representing 1, and to "open" them for himself (by predicting what Peggy would do to convince him of which bits they represent). Moreover, these blobs are indistinguishable from those genuinely obtained by Peggy.

Even with this new property, it is still not obvious that Vic cannot learn anything from the way by which Peggy opens various blobs in the process of meeting the challenges. For instance, it is *a priori* conceivable that Vic could choose his challenges in a clever way (depending on the blobs he receives from Peggy), allowing him to subsequently convince others that Ψ is satisfiable by showing them the transcript of his conversation with Peggy (even though he may still have no idea of how to satisfy the formula).

The easiest way to solve this difficulty is to add one *optional* property to the blobs: the blobs are *chameleon* if, in addition to properties (i) through (iv), they satisfy:

v) Vic can obtain blobs that cannot be distinguished from those genuinely obtained by Peggy, but that Vic can subsequently "open" as either bit, whichever she later chooses.

In other words, chameleon blobs allow Vic to do precisely what property (iii) prevents Peggy from doing. Even if Peggy and Vic have similar computing abilities, as we shall see in section 6.1, this property can sometimes still 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, without ever actually having to communicate with her. This clearly shows the minimum disclosure property of the protocol because Vic does not gain anything from running the protocol with Peggy, which he could not have obtained all by himself [GMR]. It is important to understand that this remains true even if Vic deviates arbitrarily from his prescribed behaviour. In our context, however, there is only one way in which Vic can deviate without Peggy stopping the protocol altogether: it is by choosing which challenge to issue depending on the blobs offered by Peggy, rather than doing so randomly. However, because of the properties of chameleon blobs, no such strategy is of any use to Vic.

In order to simulate one round of the protocol, Vic creates as many blobs as Peggy would have sent him, except that Vic's blobs are chameleon.

Then, Vic looks at these blobs and chooses his challenge exactly as if they actually came from Peggy. If he chooses Challenge "A", he randomly scrambles the Boolean circuit to produce something like figure 3, and he "opens" all the blobs accordingly. If he chooses Challenge "B", he randomly selects one row in each truth table and one Boolean value for each wire in the circuit (except for the value of the final output wire, which must be 1); he then "opens" the blobs in these rows to reflect the value chosen for the corresponding wires, thus producing something like figure 4. In order to simulate the whole protocol, he simply repeats the above as many times as required.

If chameleon blobs are used, the efficiency of the basic protocol can be improved because all its rounds can be conducted in parallel: Peggy sends all at once blobs corresponding to k scrambled circuits, Vic sends his string of challenges, and Peggy opens the blobs as requested by the challenges. We leave it as an exercise for the reader to see why the parallel version of our basic protocol remains just as safe if implemented with chameleon blobs.

If the blobs are *not* chameleon, it is still possible for Vic to attempt simulating one round of the protocol, except that he will fail with probability 50%. In order to do this, Vic proceeds as follows:

- he flips a coin to decide whether he will be prepared to answer Challenge "A" or Challenge "B";
- he randomly generates a figure 3 or a figure 4, depending on the outcome of the coin flip;
- he uses property (iv) to produce a sequence of blobs that he knows how to open either as figure 3 or as figure 4, whichever is the case;
- he then (honestly!) asks himself which challenge he would issue at this point if he had just received all these blobs from the real Peggy; and
- if he asked himself the challenge that he can meet, he opens the relevant blobs—otherwise he fails.

The crucial point is that property (ii) of the blobs ensures that there is no correlation between the challenge Vic decided he would be ready to meet and the challenge he actually issues to himself. In order to simulate the whole k-round protocol, Vic must repeat the above an average of $2k$ times, pretending that the unlucky rounds never happened.

This reasoning does *not* generally extend to the parallel version of the protocol when the blobs do not have the chameleon property. Consider for instance the following strategy for Vic: after receiving blobs corresponding to all k scrambled circuits, he concatenates them together (assuming each blob is given as a binary string) 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. If Vic tries to simulate this protocol, the one-way function creates a dependency between the challenges that he is ready to meet and those that he actually issues to himself, resulting in an exponentially small probability of success. Even though running the protocol with Peggy may not help Vic in learning anything about Peggy's secret, its transcript may enable him to convince someone else of the existence of Peggy's secret, because Vic could almost certainly not have produced the transcript otherwise. 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 [S]), yet it can be used to convince someone else of the secret's existence!

The proof of this section actually corresponds to a special kind of minimum disclosure, called *non-transitive minimum disclosure.* As already mentioned, such protocols do not yield Vic even the ability to later convince others of Peggy's claim. This concept is equivalent to the notion of ero-knowledge′ defined in [GMR]. The class of minimum disclosure protocols, however, includes other members, called *transitive* minimum disclosure, which do not have this property. Although they do not give Vic any computational advantage for figuring out Peggy's secret, they give him a transcript that he could not have produced by himself. Section 6.1.2 considers this idea in more detail.

4. A Complexity Theoretic Point of View

Because satisfiability of Boolean circuits is **NP**-complete [Co, GJ], the basic protocol can be used to supply minimum disclosure proofs of knowledge for any positive statement concerning a language L in **NP**. Assume without loss of generality that $L \subseteq \Sigma^*$, where Σ stands for $\{0, 1\}$ (i.e., elements of L consist of binary strings). By the definition of **NP**, there exists a "proof system" $Q \subseteq L \times \Sigma^*$ such that

- $(x \in L)(c \in \Sigma^*)[\,|c| \leqslant p(|x|) \text{ and } <x, c> \in Q\,]$
  for some fixed polynomial p, where $|x|$ denotes the length of x; and
- there exists a polynomial-time (deterministic) algorithm capable of deciding, given x and c, whether $<x, c> \in Q$.

In other words, whenever $x \in L$, there exists a succinct "certificate" c to this effect, and one can efficiently verify that c is a valid proof that $x \in L$. Using our terminology, such a c is what we called "verifiable information" to the effect that $x \in L$.

Using Cook's theorem [Co], both Peggy and Vic can efficiently build from any $x \in \Sigma^*$ a Boolean formula $\Psi_L(x)$ which is satisfiable if and only if $x \in L$. Moreover, because the proof of Cook's theorem is constructive, it is enough for Peggy to know some succinct c such that $<x, c> \in Q$ in order to efficiently deduce a satisfying assignment for $\Psi_L(x)$.

Therefore, if $L \in$ **NP**, $x \in L$ and if Peggy knows a succinct certificate c to the effect that $x \in L$, then Peggy can use the basic protocol to convince Vic that $x \in L$ and that she knows how to prove it. This is a minimum disclosure protocol assuming of course that Vic knew already the proof system for L and our basic protocol (otherwise, much information is given to Vic when Peggy instructs him about these). For most practical applications, it is better to conceive an *ad hoc* verifying circuit, rather than building it through the machinery of Cook's theorem.

As pointed out in [FFS], one may prefer not call this type of protocol "zero-knowledge" [GMR], because Vic does gain knowledge from running it — in particular the fact that $x \in L$. Following [GHY], this is why we use the word "minimum" rather that "zero": Vic learns exactly what the protocol is intended to convince him of, and nothing else.

It is interesting to consider both restrictions and extensions of **NP** in the context of minimum disclosure proofs of knowledge.

The interesting restriction concerns languages $L \in \mathbf{NP} \cap \text{co}-\mathbf{NP}$. In this case, one can construct for each $x \in \Sigma^*$ two Boolean formulae $A_L(x)$ and $B_L(x)$ such that exactly one of them is satisfiable ($A_L(x)$ if $x \in L$ and $B_L(x)$ if $x \notin L$). Clearly, their disjunction $C_L(x) = [A_L(x) \textbf{ or } B_L(x)]$ is always satisfiable. Assume now that Peggy knows whether $x \in L$ or not, and that she has the corresponding succinct **NP** certificate. This gives her a satisfying assignment for either $A_L(x)$ or $B_L(x)$, whichever is satisfiable, hence she can also satisfy $C_L(x)$. Consider what happens if she uses our basic protocol to convince Vic that she knows a satisfying assignment for $C_L(x)$. Clearly, this does not disclose anything about x to Vic (because $C_L(x)$ is always satisfiable). However, it convinces Vic that Peggy knows whether $x \in L$ or not, and that she can prove it. This issue and its applications are discussed in [FFS].

Minimum disclosure protocols can also extend beyond **NP** if we allow the certifying procedure to be probabilistic. Recall that **BPP** stands for the class of decision problems that can be solved in probabilistic polynomial time with bounded error probability [G]. It is reasonable to consider **BPP** as the *real* class of tractable problems (rather than **P**) because the error probability can always be decreased below any threshold $\epsilon > 0$ by repeating the algorithm $\alpha \log \epsilon^{-1}$ times and taking the majority answer, where the constant α depends only on the original error probability. It is generally believed that there is no inclusion relation either way between **NP** and **BPP**: nondeterminism and randomness seem to be uncomparable powers. These powers can be combined in several ways. We consider Babai's class **MA** [Ba] to be the most natural, but we prefer calling it **NBPP**. This class is such that $\mathbf{NP} \cup \mathbf{BPP} \subseteq \mathbf{NBPP}$, hence **NP** is almost certainly a strict subset of **NBPP**.

The class **NBPP** is defined exactly as **NP**, except that we are satisfied with a **BPP** algorithm for deciding, given x and c, whether $<x, c> \in Q$ (i.e., we only ask that $Q \in \mathbf{BPP}$). Whenever $<x, c> \in Q$, we now refer to c as a *convincing argument* for the fact that $x \in L$ (we no longer call it a *certificate* because it cannot be verified with certainty in general).

Section 5 shows how to obtain minimum disclosure protocols for any language L in **NBPP**. As is usual, we assume that both Peggy and Vic have "reasonable" computing power and similar algorithmic knowledge, but that Peggy is initially given a succinct convincing argument c to the effect that $x \in L$. This allows Peggy to initially convince herself beyond any reasonable doubt that $x \in L$, by running the **BPP** algorithm on input $<x, c>$. The purpose of our protocol is for Peggy to convince Vic that she knows such a c, without disclosing anything else.

The class **NBPP** is Babai's class **MA**, which he defined for his "Arthur-Merlin games" [Ba] (and similar to Papadimitriou's "stochastic satisfiability" in his "games against nature" [Pa]). According to Babai, his other class **AM** is a better candidate for the generalization of **NP** to probabilistic computations. In particular, he proved that **MA** $\subseteq$ **AM**. The interest in **AM** is further increased by the proof that, for any fixed $k \geqslant 2$, **AM** = **IP**(k), the class of languages that allow an interactive protocol with no more than k rounds [GS]. All these considerations are theoretically very compelling.

We claim nonetheless that **MA** (i.e., **NBPP**) is a more *natural* class for *practical* purposes, at least in cryptographic settings. If $L \in$ **NBPP** and $x \in L$, it is enough for Peggy to know one succinct convincing argument c to this effect. In practice, it is not necessary that this c be God-given to Peggy. As mentioned earlier, it is easy for some languages to generate both x and the corresponding c by a probabilistic process. Consider for instance the set B of integers having exactly two prime factors. If Peggy generates two distinct random integers p and q that pass a probabilistic primality test [SS, R1] to her satisfaction, she is convinced that $n = pq$ is a member of B and her convincing argument is $<p, q>$. The protocol given in section 5 allows her to convince Vic that $n \in B$ without disclosing anything else. (In this case, $B \in$ **NP** because the set of primes is in **NP** [Pr]. This does not reduce the practical interest of our example, however, because Peggy may find it prohibitive to convert her **NBPP** convincing argument $<p, q>$ into an **NP** certificate $<p, c(p), q, c(q)>$, where $c(\bullet)$ stands for an **NP** certificate that $\bullet$ is prime. This remark remains true in practice despite the results of [GK, AH].)

By contrast, it is not clear that Peggy can reasonably be asked to carry out an **AM** protocol (regardless of minimum disclosure considerations), even if she were initially given a succinct piece of advice (because an **AM** protocol would in general require Peggy to determine an **NP**-like certificate as a function of a random string supplied by Vic). This is a pity in some sense because it is trivial that our basic protocol allows the transformation of any **AM** protocol into a minimum disclosure one. As explained in [GMW], this is true because once "Arthur" has given "Merlin" his coin flips, it "only" remains for Merlin to satisfy an **NP** statement, which can be done without disclosing anything else if the basic protocol of section 2 is used. For more details on how an infinitely powerful prover can carry out a "zero-knowledge" proof for any language that admits an interactive protocol in the sense of [GMR] (a result independently obtained by Ben-Or and Impagliazzo), consult [GMW].

5. Going Beyond NP: the Probabilistic Case

Consider any language $L \in \mathbf{NBPP}$. Let x be such that Peggy knows a succinct convincing argument c to the effect that $x \in L$. Because c is not an **NP** certificate, Peggy is not absolutely certain that $x \in L$, but she can reduce her probability of error below any desired threshold by the virtues of **BPP**. The purpose of the minimum disclosure protocol described in this section is for Peggy to convince Vic that $x \in L$ and that she knows a convincing argument to this effect. We call this process a "*non-transitive transfer of confidence*" because it convinces Vic that $x \in L$ (a statement about which Peggy is convinced already) in a way that he cannot subsequently convince anyone else.

Note that Vic could be fooled by this process in several different ways. It can be that Peggy is dishonest and that she does not really know a convincing argument to the effect that $x \in L$, but that she succeeds (with exponentially small probability) in fooling Vic by being lucky enough to be asked each time the only challenge she is capable of answering (exactly as she could have done with the basic deterministic protocol). It is also possible that Peggy is honest but wrong in her belief (because the certifying **BPP** algorithm incorrectly lead Peggy to believe that her convincing argument was correct). In this case, it is most likely that Peggy will discover her mistake as a result of trying to convince Vic, but it is also possible that the certifying algorithm will err once more. Finally, it is possible that Peggy is honest and correct in her claim, but that when she runs the protocol with Vic the verdict comes out wrong due to an error of the certifying algorithm.

As a preliminary step, Peggy and Vic agree on the error probability ϵ they are willing to tolerate for the certifying algorithm. From this agreement, they modify the algorithm so that its probability of error does not exceed ϵ (for this, they first determine how many times the original algorithm must be repeated so that the majority answer is almost certainly correct). From now on, we assume without loss of generality that the probability of error of the certifying algorithm is negligible.

Intuitively, Peggy wants to convince Vic that she knows some secret input c such that the certifying algorithm will (almost certainly) accept the input $<x, c>$. Let n and m denote the size of x and c, respectively. Assume for simplicity that the value of m is uniquely determined as a known (easy to compute) function of n, so that there is no need for the protocol to hide the value of m from Vic. Let r be an upper bound on the number of coin flips that the certifying algorithm can perform on any input $<x, \hat{c}>$, where $\hat{c}$ is of size m. An argument similar to the proof of Cook's theorem shows that this gives rise to a Boolean formula Ψ with at least $m + r$ variables so that if the first m variables are set to represent the binary string c and the next r variables are determined by independent random coin flips, then (except with probability at most ϵ) it is easy to set the other variables (if any) so as to satisfy the whole formula if and only if c is a valid convincing argument to the effect that $x \in L$. This formula can be constructed from knowledge of x and of the certifying procedure. As before, it is converted into a Boolean circuit on which both Peggy and Vic agree.

The basic minimum disclosure protocol of section 2 cannot be used directly because Vic cannot trust Peggy to choose the r appropriate inputs truly at random. On the other hand, Peggy cannot allow Vic either to choose these variables, because a careful choice might allow Vic to obtain information on Peggy's secret convincing argument c. It is therefore necessary to use a coin flipping sub-protocol to set these inputs to random values not under the control of either party. Moreover, Vic should not be allowed to see the outcome of the coin flips, again to prevent him from learning information about c (i.e., coin flipping should be performed "in a well" [Bl]). Finally, Peggy must not be allowed to choose her c as a function of the coin flips.

It is much easier to implement all these requirements if we ask that blobs have two additional properties. As we shall see in section 6.4, however, property (vii) is always a consequence of property (vi), and property (vi) is always a consequence of properties (i) through (iv):

vi) Given two blobs, Peggy can convince Vic that they represent the same bit (if this is so) without disclosing any information about which bit this is.

vii) Given two blobs, Peggy can convince Vic that they represent distinct bits (if this is so) without disclosing any information as to which one represents 0 and which one represents 1.

Coin flipping in a well is obvious to implement with property (vii): Peggy generates two blobs representing distinct bits, she convinces Vic that this is so, and she asks him to choose one of them. When Vic makes his choice, the coin flip is determined and Peggy knows its outcome, but Vic cannot tell how it went unless Peggy subsequently opens the blob he chose (which she will *never* do in the protocol below). Property (ii) prevents Vic from influencing the coin flip, and properties (iii) and (vii) prevent Peggy from doing so.

We are now ready to describe our protocol for non-transitive transfer of confidence. Recall that Peggy and Vic have agreed on a Boolean circuit corresponding to the certifying algorithm intended to probabilistically verify Peggy's secret convincing argument c to the effect that $x \in L$. At this point, Peggy commits once and for all to her convincing argument by giving m blobs to Vic corresponding to the bits of c. Then, Peggy and Vic flip r coins in the well by the above procedure, which results in r blobs corresponding to the outcome of the coin flips. It only remains for Peggy to a use a slight variation on the basic protocol of section 2 to convince Vic that she knows how to select the other inputs of the circuit (if any) so as to satisfy it.

The basic protocol must be modified in order to force Peggy to use the proper bits for the inputs corresponding to c and to the the coin flips, but this must be achieved without disclosing anything that could help Vic learn about the value of these bits. We illustrate how this can be done with the example of section 2. Assume that Peggy has given some blob b to Vic (which represents the bit 1, but Vic does not know this). Peggy wishes to convince Vic that she knows a satisfying assignment for Ψ in which the first input corresponds to the bit represented by blob b. For this purpose, Peggy scrambles the Boolean circuit to produce a figure 3, and she gives it to Vic after encoding it with blobs, exactly as before. If Vic issues Challenge "A", she opens each and every blob to show figure 3 to Vic, still exactly as in the deterministic basic protocol. If Vic issues Challenge "B", however, she must do more than showing figure 4 to Vic (which would say nothing about the first Boolean variable of the satisfying assignment). Because the wire corresponding to the first input is set to 0 in figure 4 (as shown by the first bit in the outlined row of the top left truth table), Peggy uses property (vi) to convince Vic that the blob associated with the corresponding wire complementation represents the same bit as blob b. If the wire corresponding to this input had been a 1, she would of course have used property (vii) instead.

This completes the description of our protocol for non-transitive transfer of confidence. In general, however, it is *not* a minimum disclosure protocol from a theoretical point of view. The subtle difficulty is that different convincing arguments may cause the certifying procedure to fail with different probabilities. Because he is generally unable to predict the failure probability, Vic cannot simulate exactly the conversation that would take place if he were really talking to Peggy. Moreover, running the protocol an exponentially large number of times with Peggy could in principle allow a very powerful Vic to increase his chances to guess correctly Peggy's secret (by keeping a tally of how many times the protocol showed a failure of the certifying algorithm). For all practical purposes, however, this threat is of no consequence if ϵ is chosen small enough, and the protocol can thus be used safely.

A variation on this scheme *is* almost always minimum disclosure, but it is usually more time consuming. To achieve this, the original **BPP** certifying algorithm has to be modified, by repeating it enough times and taking the majority, so that all but exponentially few of the random choices may cause it to give the wrong answer on even a single input. The fact that this is possible can be proven by a refinement of the proof that $\mathbf{MA} \subseteq \mathbf{AM}$ [Ba]. This allows use of the basic protocol from section 2 almost directly, with no need for the coin tossing to be in a well or for blob properties (vi) and (vii).

6. Blob Implementations

In the previous sections we have taken the existence of blobs for granted. 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 therefore be based on the particular requirements of the application. The safety of most of the following implementations depends on unproved assumptions about the computational difficulty of solving particular problems. Section 7 compares the advantages and drawbacks of these various approaches.

As an elementary (but probably not very secure) example, consider two isomorphic graphs G and H. Assume that Peggy is convinced that they are isomorphic, but that she does not actually know an isomorphism between them. Suppose further that she is computationally incapable of finding one such isomorphism in a reasonable amount of time. (Let us ignore for the moment the question of how Peggy could be convinced that the graphs are isomorphic without explicitly knowing an isomorphism.) In this setting, Peggy agrees with Vic that any graph for which she can show an isomorphism with G (resp. H) represents the bit 0 (resp. 1). Referring to the defining properties of blobs given in the introduction and in section 3, it is clear that properties (i) and (iv) hold because both Peggy and Vic can produce blobs representing 0 (resp. 1) by randomly permuting the vertices of G (resp. H). Property (ii) is also unconditionally true because blobs representing 0 are information theoretically indistinguishable from blobs representing 1. In order to open a blob, it suffices for Peggy to show Vic the isomorphism she knows with G or H, whichever is the case. However, this implementation allows Peggy to cheat to her heart's contents if she has an efficient algorithm for figuring out graph isomorphisms. Under the assumption that she is incapable of finding the isomorphism between G and H while the protocol is going on, however, property (iii) holds and thus the protocol is safe. Notice that it does Peggy no good at all to figure out an isomorphism between G and H after the protocol is completed. When Vic knows the isomorphism, the blobs are cameleon. But the protocol is secure wether or not Vic knows the isomorphism.

As illustrated by this example, blobs really consist of two different parts, which are used at different times: the "visible" part issued initially by Peggy to force her to commit to a specific bit (some graph isomorphic to both G and H) and the "secret" part by which she may subsequently open the blob (the actual isomorphism known by Peggy between this graph and either G and H). Thus, most *bit commitment schemes* (but section 6.3 gives exceptions) consist of two sets X and Y together with an efficiently computable *verification function* $v : X \times Y \to \{0, 1, \bullet\}$, where "$\bullet$" stands for "undefined", such that:

i) Given $b \in \{0, 1\}$, Peggy can easily generate several pairs $x \in X$ and $y \in Y$ such that $v(x, y) = b$.

ii) When Peggy gives Vic some $x \in X$, Vic cannot tell whether Peggy also knows a $y \in Y$ such that $v(x, y) = 0$ or such that $v(x, y) = 1$.

iii) Peggy opens a blob (x, y) revealing y; Peggy cannot obtain any triple $x \in X$, $y_1 \in Y$ and $y_2 \in Y$ such that $v(x, y_1) = 0$ and $v(x, y_2) = 1$.

iv) Given $b \in \{0, 1\}$, Vic can also generate pairs $x \in X$ and $y \in Y$ such that $v(x, y) = b$. Moreover, Vic can generate these pairs with exactly the same probability distribution as Peggy.

These four requirements are slight restrictions on the corresponding defining blob properties, which is why we continue to refer to them by the same symbols. The elements of X correspond to the visible part of the blob and the elements of Y correspond to its secret part.

There is an apparent contradiction between properties (ii) and (iii). If there exist x, y_1 and y_2 such that $\nu(x,y_1)=0$ and $\nu(x,y_2)=1$, why should Peggy be unable to obtain them (and thus violate property (iii))? On the other hand, if each $x \in X$ unambiguously determines the only possible non-undefined value for $\nu(x,y)$, why should Vic not be able to determine this value upon seeing x (and thus violate property (ii) by learning the bits represented by Peggy's blobs)?

We offer three different ways to resolve this difficulty. Section 6.1 investigates blobs that are *unconditionally secure for* Peggy. In this case, property (ii) holds regardless of Vic's computing power and algorithmic knowledge. This is achieved by asking that, for every x in X, there must exist at least one y_1 and one y_2 in Y such that $\nu(x,y_1)=0$ and $\nu(x,y_2)=1$. Moreover, the probability that Peggy generates any given x when he uses property (i) to obtain a blob must be the same whether she wishes to represent 0 or 1. These additional requirements clearly imply that Vic cannot learn anything about the bit represented by a blob unopened by Peggy. However, they also imply that Peggy could in principle violate property (iii), but our implementations are designed to make this computationally infeasible for her (under suitable assumptions). The graph isomorphism implementation outlined above fits in this category. This approach leads to protocols similar to those of [Ch3, BC2]. Some of the implementations are also chameleon, which therefore allows the basic protocol to be carried out in parallel, as explained in section 3.

Section 6.2 investigates blobs that are *unconditionally secure for* Vic. In this case, property (iii) holds regardless of Peggy's computing power and algorithmic knowledge. This is achieved by asking that, for every x in X, there must *not* exist simultaneously a y_1 and a y_2 in Y such that $\nu(x,y_1)=0$ and $\nu(x,y_2)=1$. This additional requirement clearly implies that Peggy is irrevocably committed to one bit or the other each time she utters the visible part of a blob. However, it also implies that Vic could in principle violate property (ii), but our implementations are designed to make this computationally infeasible for him (under suitable assumptions). This approach leads to "zero-knowledge interactive protocols" as defined by [GMR] and to protocols similar to those of [GMW, BC1].

Section 6.3 investigates blobs that would not be weakened even if all parties considered had unlimited computing power. Blobs of section 6.3.1 make use of quantum physical principles. Using these blobs, it is provably impossible for Vic to obtain any information on Peggy's secret (assuming that quantum physics is correct). Although quantum blobs could be cheated in principle by Peggy, this would require a technology far beyond any foreseeable future. Blobs of section 6.3.2 can be used in a multiparty environment under the assumption that the honest participants outnumber the cheaters by a constant ratio.

Finally, section 6.4 shows how to transform any bit commitment scheme into a scheme that also allows Peggy, given the visible part of two blobs, to convince Vic of whether they represent the same bit or not, whichever is the case, without disclosing anything else. This possibility was used extensively in section 5 (properties (vi) and (vii)). As a corollary, this yields a protocol to achieve "coin-tossing in a well" [Bl], assuming only that secure probabilistic encryption schemes exist in the sense of [GM].

6.1. Blobs unconditionally secure for Peggy

6.1.1. Based on factoring [BC2]

Only a little elementary number theory is necessary to understand this particular implementation of blobs. Let n be an integer. $\mathbb{Z}_n^*$ denotes the set of integers relatively prime to n between 1 and $n-1$. An integer $x \in \mathbb{Z}_n^*$ is a *quadratic residue* modulo n ($x \in \mathrm{QR}_n$) if there exists a $y \in \mathbb{Z}_n^*$ such that $x \equiv y^2 \pmod{n}$. Such a y is called a *square root* of x, modulo n. Let s be any fixed quadratic residue. A uniformly distributed random quadratic residue can be generated by choosing $y \in \mathbb{Z}_n^*$ at random and computing $x = y^2 s \bmod n$. This holds in particular if $s = 1$. The crucial fact is that it is *information theoretically* impossible to distinguish a quadratic residue thus produced using any given $s \in \mathrm{QR}_n$ from one produced using $s = 1$.

Now, let $n = pq$ be the product of two distinct odd primes. The problem of extracting square roots modulo n is computationally equivalent to the problem of factoring n [R2]. We shall assume here that factoring n is almost always infeasible when p and q are sufficiently large. Therefore, given n and $s \in \mathrm{QR}_n$, we assume that it is infeasible to compute a square root of s modulo n without the factorization of n.

At the outset of the protocol, Vic randomly chooses two distinct large primes p and q, and he computes their product $n = pq$. Vic also chooses a random $t \in \mathbb{Z}_n^*$ ($t \neq \pm 1$) and computes $s = t^2 \bmod n$. Vic gives n and s to Peggy. Using a minimum disclosure protocol [Be, BC1], Vic convinces Peggy that s is a quadratic residue modulo n and that he knows one of its square roots. (Notice that in this initialization phase of the protocol, it is temporarily Vic that takes the role of Prover and Peggy that of Verifier.)

The blobs are now defined by the sets $X = \mathrm{QR}_n$, $Y = \mathbb{Z}_n^*$ and

$$\nu(x,y) = \begin{cases} 0 & \text{if } x \equiv y^2 \pmod{n} \\ 1 & \text{if } x \equiv y^2 s \pmod{n} \\ \bullet & \text{otherwise}. \end{cases}$$

Requirement (i) holds because whenever she wishes to represent some bit b by a blob, Peggy randomly chooses a $y \in \mathbb{Z}_n^*$ and computes $x = y^2 s^b \bmod n$. She gives x to Vic but she keeps y secret as her *witness* to the effect that the blob x represents bit b. Clearly, any quadratic residue can represent 0 just as well as 1, depending only on Peggy's knowledge about it. Therefore, requirement (ii) holds in a very strong sense: blobs convey *no* information on the bits they represent. Requirement (iii) holds because Peggy could easily compute a square root of s (which we assumed to be computationally infeasible for her) from knowledge of y_1 and y_2 such that $y_1{}^2 \equiv y_2{}^2 s \pmod{n}$, and she cannot have the choice of opening a blob as either 0 or 1 without this knowledge. Requirement (iv) holds because Vic can create blobs in exactly the same way as Peggy does.

It is obvious that these blobs leave the door wide open for Peggy to cheat if she succeeds in extracting a square root of s. It is a more subtle observation that there is also a possibility for Vic to cheat and thus learn everything about Peggy's secret. In order to achieve this, Vic must be daring from the beginning, because he must give Peggy a quadratic *non*-residue as his s. If he succeeds in convincing Peggy that s is a quadratic residue (which can only happen with an exponentially small probability, regardless of Vic's computing power), then blobs that Peggy will give him in the future will represent 0 if and only if they are quadratic residues, a condition that Vic can easily determine with the help of his factoriation zof n.

In addition, these blobs are chameleon (property (v) from section 3) because Vic's knowledge of t, a square root of s, allows him to create blobs that he can open in either way. For this purpose, Vic generates a random $y \in \mathbb{Z}_n^*$ and computes $x = y^2 s \bmod n$. In order to simulate Peggy's opening of this blob as a 1, Vic pretends that Peggy showed him y; in order to simulate Peggy's opening of this blob as a 0, Vic computes $\hat{y} = yt \bmod n$ and pretends that Peggy showed him $\hat{y}$.

6.1.2. Based on the discrete logarithm [CDG]

Let p be a large prime and let α be a generator of $\mathbb{Z}_p^*$, 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 [DH]. It can be used also to create blobs as follows.

At the outset of the protocol, Peggy and Vic agree on a specific choice of p and α. They are both absolutely certain (not just convinced) that p is a prime and that α is a generator of $\mathbb{Z}_p^*$ (perhaps because they know the factors of $p-1$). These same parameters can be public, in the sense that the same $<p, \alpha>$ can be used with no (known) breach of security by all people wishing to engage in minimum disclosure protocols. At the outset, Vic also chooses a random $s \in \mathbb{Z}_p^*$ $(s \neq 1)$, which he gives to Peggy. Assuming the intactability of the discrete logarithm, Peggy cannot compute in a reasonable amount of time the unique e such that $1 \leqslant e \leqslant p-2$ and $s \equiv \alpha^e \pmod p$.

The blobs are now defined by the sets $X = \mathbb{Z}_p^*$, $Y = \{0, 1, 2, \ldots, p-2\}$ and

$$\nu(x,y) = \begin{cases} 0 & \text{if } x \equiv \alpha^y \pmod p \\ 1 & \text{if } x \equiv s\alpha^y \pmod p \\ \bullet & \text{otherwise} . \end{cases}$$

Requirement (i) holds because whenever she wishes to represent bit b by a blob, Peggy randomly chooses a $y \in Y$ and she computes $x = s^b \alpha^y \textbf{ mod } p$. She gives x to Vic but keeps y secret as her witness to the effect that blob x represents bit b. Clearly, any element of $\mathbb{Z}_p^*$ can just as well represent a 0 as a 1, depending only on Peggy's knowledge about it. Therefore, requirement (ii) holds unconditionally, as with the implementation of section 6.1.1; blobs still contain no information on the bits they represent. Requirement (iii) holds because Peggy could easily compute e (which we assumed to be computationally infeasible for her) from knowledge of y_1 and y_2 such that $\alpha^{y_1} \equiv s\alpha^{y_2} \pmod p$, and she cannot have the choice of opening a blob as either 0 or 1 without this knowledge. Requirement (iv) holds because Vic can create blobs in exactly the same way Peggy does.

Despite a superficial resemblance, there is a fundamental difference between this implementation of blobs and that of section 6.1.1. There is no longer any possibility for Vic to cheat. The fact that blobs representing 0 and blobs representing 1 are information theoretically indistinguishable depends only on the fact that p is a prime and that α generates $\mathbb{Z}_p^*$, and both these conditions were thoroughly verified by Peggy before starting the protocol. Using the terminology of [GMW], this implementation of blobs turns the basic protocol into a "perfect zero-knowledge interactive protocol" for satisfiability (except that it does not fit their model as an interactive protocol since they allow Peggy to be infinitely powerful, in which case she would have no problem computing e — this "unfortunately" prevents Fortnow's result [F] to apply). Such a perfect zero-knowledge interactive protocol was *incorrectly* claimed in [BC2] about the implementation corresponding to the blobs of section 6.1.1. Notice however that it is computationally more efficient to use the blobs of section 6.1.1.

Besides efficiency, there is another price to pay for this advantage: the "discrete logarithm blobs" are not chameleon, and thus the basic protocol should not directly be performed in parallel. If it were, Vic could cheat by choosing a random integer e and computing $2\alpha^e \textbf{ mod } p$ 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. This displays 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 [S]!

With some preprocessing, it is possible to add the chameleon property to these blobs. Rather than choosing s randomly in $\mathbb{Z}_p^*$, Vic randomly chooses an integer e between 1 and $p-2$ and computes $s = \alpha^e \ \mathbf{mod}\, p$. Of course, Peggy should only be willing to conduct the minimum disclosure protocol in parallel after Vic has convinced her (via a minimum disclosure protocol [CEGP]) that he knows the discrete logarithm of s.

6.1.3. Based on graph isomorphism [BC2]

Define a graph G to be *hard* if it is computationally infeasible with high probability, given G and a random isomorphic copy of G, to figure out the isomorphism. Let us assume that hard graphs exist and that they can be obtained in practice.

At the outset of the protocol, Peggy and Vic agree on some hard graph $G = <N, E>$. Vic randomly chooses a permutation $\sigma : N \to N$ and uses it to produce $H = <N, E'>$, where $(u, v) \in E'$ if and only if $(\sigma(u), \sigma(v)) \in E$. He then gives H to Peggy and convinces her that G and H are isomorphic without disclosing anything about the isomorphism σ [GMW]. By our assumption, it is computationally infeasible for Peggy to determine σ (or any other isomorphism between G and H) in a reasonable amount of time.

The blobs are now defined by the sets $X = \{ K = <N, \hat{E}> \mid K \text{ is a graph isomorphic to } G \}$, $Y = \{ \gamma : N \to N \mid \gamma \text{ is a permutation} \}$ and

$$\nu(<N, \hat{E}>, \gamma) = \begin{cases} 0 & \text{if } (u, v) \in \hat{E} \text{ iff } (\gamma(u), \gamma(v)) \in E \\ 1 & \text{if } (u, v) \in \hat{E} \text{ iff } (\gamma(u), \gamma(v)) \in E' \\ \bullet & \text{otherwise} . \end{cases}$$

The reader can easily verify that properties (i) through (iv) hold, and that these blobs are also chameleon.

6.2. Blobs unconditionally secure for Vic

6.2.1. Based on quadratic residuosity [BC1]

To understand these blobs, more elementary number theory is needed. We refer the reader to [BC1] for the relevant background.

At the outset of the protocol, Peggy randomly chooses two distinct large primes p and q, and she computes their product $n = pq$. She also randomly chooses a quadratic non-residue s modulo n with Jacobi symbol $+1$. She discloses n and s to Vic. She then convinces Vic that n has only two prime factors and that s is a quadratic non-residue modulo n, without disclosing any additional information [GMR, GHY]. Following the Quadratic Residuosity Assumption [GM], we assume that Vic cannot distinguish random quadratic residues from non-residues with Jacobi symbol $+1$.

The blobs are now defined by the sets $X = \mathbb{Z}_n^*[+1]$, $Y = \mathbb{Z}_n^*$, and

$$v(x,y) = \begin{cases} 0 & \text{if } x \equiv y^2 \pmod{n} \\ 1 & \text{if } x \equiv y^2 s \pmod{n} \\ \bullet & \text{otherwise}. \end{cases}$$

Requirement (i) holds because whenever she wishes to represent some bit b by a blob, Peggy randomly chooses a $y \in \mathbb{Z}_n^*$ and she computes $x = y^2 s^b \bmod n$. Clearly, x represents 0 if and only if it is a quadratic residue, a condition that we have assumed infeasible for Vic to test. This shows that requirement (ii) holds from a computational point of view. Requirement (iii), however, holds unconditionally because any given x is either a quadratic residue or not. Requirement (iv) holds because Vic can create blobs exactly the same way Peggy does.

6.2.2. Based on probabilistic encryption schemes in general [GMW]

A probabilistic encryption scheme in the sense of [GM] is a polynomial-time computable function $f: B \times Y \to X$ that, on input $b \in B$ and "coin tosses" $y \in Y$, outputs an encryption $f(b,y)$ of b. Decryption is unique: $f(b,y) = f(\hat{b},\hat{y})$ implies that $b = \hat{b}$. However, it is assumed to be computationally infeasible to determine b (or obtain any information about it) from $f(b,y)$ without knowledge of some "trap-door" information.

Taking $B = \{0, 1\}$, we define blobs by

$$v(x,y) = \begin{cases} 0 & \text{if } f(0,y) = x \\ 1 & \text{if } f(1,y) = x \\ \bullet & \text{otherwise}. \end{cases}$$

The reader can easily check that all four blob requirements are verified.

6.3. Blobs that no amount of computing power can break

6.3.1. Quantum blobs [BB3]

We assume in this section that the reader is familiar with the principles of Quantum Cryptography [BB1, BB2]. Charles H. Bennett has suggested that blobs could be implemented with similar principles. Indeed, the use of quantum physics leads to an implementation of blobs that is very similar to the paper-and-envelope "implementation" given as an illustration in the introduction. However, it removes the potential threats of Vic reading the blob through a sealed envelope and of Peggy changing the contents of the envelope after Vic has applied his signature to it. Quantum blobs are implemented by a process identical to quantum coin-tossing [BB2], which we do not repeat here. Let us only say that it can be proven that any cheating successfully conducted by Vic would lead to an apparatus capable of transmitting information faster than the speed of light. In principle, the Einstein-Rosen-Podolsky "paradox" [EPR, M] allows Peggy to cheat, exactly as with the coin-tossing protocol, but the technology needed to perform this cheating is far beyond any foreseeable future. More details are forthcoming [BB3].

6.3.2. Multi-party blobs [CCD]

Unconditionally secure blobs can be obtained in a multi-party environment, only assuming that more than two thirds of the participants are honest and that each pair of participants shares a private channel. Even a coalition of nearly a third of the participants with unlimited computing power cannot cheat the honest ones. For more details, consult [CCD].

6.4. Proving blob equality and inequality

In this section, we address the question of how Peggy can convince Vic, given any two blobs x and $\hat{x}$, that they represent the same bit or alternatively that they represent opposite bits, without disclosing anything else. Referring to section 5, we distinguish between properties (vi), which allows Peggy to convince Vic that two blobs represent the same bit (when in fact they do), and property (vii), which allows Peggy to convince Vic that two blobs represent distinct bits (when in fact they do).

Most specific implementations of blobs given above (sections 6.1.1, 6.1.2, 6.1.3 and 6.2.1) allow property (vi) to be obtained as a primitive operation. Consider the blobs of section 6.1.1 or 6.2.1, for instance. If $x \equiv y^2 s^b \pmod{n}$ and $\hat{x} \equiv \hat{y}^2 s^b \pmod{n}$ for the same bit b, then if Peggy computes $z = y\hat{y}s^b \textbf{ mod } n$ and gives it to Vic, the latter will be convinced that x and $\hat{x}$ represent the same bit after checking that $x\hat{x} \equiv z^2 \pmod{n}$. However, if we use a general probabilistic encryption scheme to implement blobs (section 6.2.2), it is not clear that property (vi) is always so easy to obtain. We challenge the reader, for instance, to figure out how blob equality can be achieved with paper-and-envelope blobs or with quantum blobs.

Although property (vii) is also easy to implement as a primitive operation with the blobs of sections 6.1.1, 6.1.2 and 6.2.1, it is intriguing to notice that this does not seem to be so with the blobs based on graph isomorphism (section 6.1.3). This shows that there is a fundamental difference between properties (vi) and (vii): it is easy for Peggy to convince Vic that she knows an isomorphism between two graphs when this is so, but how could she possibly convince him that she does *not* know such an isomorphism? (Notice that this question has nothing to do with whether or not graph non-isomorphism is in **NP**.)

Even though they may not always be achieved as primitive operations, it turns out that properties (vi) and (vii) can always be obtained through an interactive sub-protocol. For simplicity, let us assume for the moment that our blobs are "mathematical" in the sense that they are described by sets of integers X and Y, and by a verification function ν (we thus temporarily rule out quantum blobs and paper-and-envelope blobs). Ivan Damgård has pointed out that the basic protocol of section 2 can be used for this purpose. Assume for instance that blobs x and $\hat{x}$ represent the same bit and that Peggy would show y and $\hat{y}$ to Vic if she wanted to open them. Instead, she uses the basic protocol to convince Vic that she knows y and $\hat{y}$ such that $\nu(x,y) = \nu(\hat{x},\hat{y}) \in \{0,1\}$, which is an **NP** statement!

For blobs that already offer property (vi) as a primitive, but not property (vii) (such as the blobs based on graph isomorphism of section 6.1.3), it is more efficient to implement property (vii) through a sub-protocol that makes use of property (vi). Let x and $\hat{x}$ be two blobs that represent distinct bits. To convince Vic of this fact, Peggy gives him two more blobs z and $\hat{z}$, claiming that they also represent distinct bits. At this point, Vic issues one of two possible challenges. As a result, Peggy must either open both z and $\hat{z}$, thus showing that one of them represents 0 and the other 1, or she must use twice property (vi) to convince Vic of the equivalence between x and z (or $\hat{z}$, whichever is the case) and the equivalence between $\hat{x}$ and $\hat{z}$ (or z). If the above is repeated k times and in fact x and $\hat{x}$ represent the same bit, Peggy has only a probability 2^{-k} of successful cheating.

The reverse process is obvious: if Peggy knows how to convince Vic that two blobs represent distinct bits when this is so, and if she wishes to convince him that blobs x and $\hat{x}$ represent the same bit, she simply gives him the proper blob x' and she convinces him that x and x' represent distinct bits and that $\hat{x}$ and x' also represent distinct bits.

For theoretical completeness, let us finally mention that Charles H. Bennett has found a way to transform any bit commitment scheme into one that also has properties (vi) and (vii): it is enough to assume the abstract properties (i), (ii) and (iii) of the introduction and property (iv) of section 3. This construction will be described in the forthcoming paper on quantum cryptography [BB3].

7. 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 is *retroactive* (or *off-line*) if it can take place some time after the protocol is completed, by looking back at its transcript. Conversely, it is *real-time* if it must be completed while the protocol is taking place.

If the blobs of section 6.2 are used (corresponding to the protocols previously given by [GMW, BC1]), 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 assumptions turned 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 the blobs of section 6.1 are used (corresponding to the protocols previously given in [Ch3, BC2]), the *only* way Vic can hope to learn anything about Peggy's secret is to be daring right from the beginning and to choose a quadratic non-residue as his s. He would almost certainly get caught by Peggy while trying to convince her that s is a quadratic residue but would if successful be capable of distinguishing Peggy's encryptions of 0 from her encryptions of 1. Asking Vic to disclose a square root of s at the very end of the protocol (which is not detrimental to him at that point, assuming he is honest), provides Peggy with *certainty* that Vic has not learned any of her secrets (and *never* will retroactively). If the blobs of section 6.1.2 are used, even this unlikely opportunity for daring cheating is not available to Vic. On the other hand, Vic's belief that Peggy cannot cheat depends on his belief in the appropriate cryptographic assumption. With the implementation of section 6.1.1, for instance, Peggy could clearly "open" any quadratic residue as either 0 or 1, whichever suits her best, if she could only obtain a square root of s *in real-time*. Moreover, even if factoring remains impractical, Peggy still has a (very slight) possibility of lucky (hence undetectable) cheating. Finally, retroactive cheating is not possible for Peggy because it is meaningless. An algorithm capable of factoring in two weeks, for instance, would spell doom to the protocol if blobs were implemented as in section 6.2.1, but it would be of no direct consequence with the blobs of section 6.1.1.

If blobs unconditionally secure for Peggy are used (section 6.1), additional security for Vic can be obtained by asking Peggy to repeat the entire protocol using a different type of blob each time. In order to cheat, this would force Peggy to be capable of breaking several different cryptographic assumptions. She would need efficient algorithms both for factoring and for extracting discrete logarithms, for instance. Curiously, the opposite effect is obtained with the blobs that are unconditionally secure for Vic (section 6.2): repeating the protocol with different types of blobs would only make it *easier* for Vic to cheat since she can do so by breaking any one of the underlying cryptographic assumptions.

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 blobs of section 6.1.1 to convince a skeptical Vic of this. At the end of the protocol, regardless of any unproved assumptions, Vic will be convinced that Peggy either has a proof of **T** or that she has hot results on integer factoring! In particular, no assumptions are needed if **T**'s claim is: "I have an efficient factoring algorithm"...

8. Acknowledgement

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

9. References

[AH] Adleman, L. M. and M.-D. A. Huang, "Recognizing primes in random polynomial time", *Proceedings of the 19th Annual ACM Symposium on the Theory of Computing*, 1987, pp. 462-469.

[Ba] Babai, L., "Trading group theory for randomness", *Proceedings of the 17th Annual ACM Symposium on the Theory of Computing*, 1985, pp. 421-429.

[Be] Benaloh (Cohen), J. D., "Cryptographic capsules: a disjunctive primitive for interactive protocols", *Advances in Cryptology: Proceedings of CRYPTO 86*, Santa Barbara, California, August 1986, Springer-Verlag, to appear.

[BB1] Bennett, C. H. and G. Brassard, "An update on quantum cryptography", *Advances in Cryptology: Proceedings of CRYPTO 84*, Santa Barbara, California, August 1984, Springer-Verlag, 1985, pp. 474-480.

[BB2] Bennett, C. H. and G. Brassard, "Quantum cryptography: public-key distribution, and coin-tossing", *IEEE International Conference on Computers, Systems and Signal Processing*, Bangalore, India, December 1984, pp. 175-179.

[BB3] Bennett, C. H. and G. Brassard, "Quantum cryptography", *in preparation.*

[Bl] Blum, M., "Coin flipping by telephone: a protocol for solving impossible problems", *Proceedings of the 24th IEEE Compcon*, 1982, pp. 133-137; reprinted in *SIGACT/NEWS*, Vol. 15, 1983, pp. 23-27.

[BC1] Brassard, G. and C. Crépeau, "Zero-knowledge simulation of Boolean circuits", *Advances in Cryptology: Proceedings of CRYPTO*

86, Santa Barbara, California, August 1986, Springer-Verlag, to appear.

[BC2] Brassard, G. and C. Crépeau, "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*, 1986, pp. 188-195.

[Ch1] Chaum, D., "Showing credentials without identification: signatures transferred between unconditionally unlinkable pseudonyms", Presented at EUROCRYPT 85, Linz, Austria, April 1985.

[Ch2] Chaum, D., "Security without identification: transaction systems to make Big Brother obsolete", *Communications of the ACM*, Vol. 28, no. 10, October 1985, pp. 1030-1044.

[Ch3] Chaum, D., "Demonstrating that a public predicate can be satisfied without revealing any information about how", *Advances in Cryptology: Proceedings of CRYPTO 86*, Santa Barbara, California, August 1986, Springer-Verlag, to appear.

[CCD] Chaum, D., C. Crépeau and I. B. Damgård, "Fundamental primitives for multiparty unconditionally-secure protocols", *in preparation.*

[CDG] Chaum, D., I. B. Damgård and 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, August 1987, Springer-Verlag, to appear.

[CEGP] Chaum, D., J.-H. Evertse, J. van de Graaf and R. Peralta, "Demonstrating possession of a discrete logarithm without revealing it", *Advances in Cryptology: Proceedings of CRYPTO 86*, Santa Barbara, California, August 1986, Springer-Verlag, to appear.

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

[DH] Diffie, W. and M. E. Hellman, "New directions in cryptography", *IEEE Transactions on Information Theory*, Vol. IT-22, 1976, pp. 644-654.

[EPR] Einstein, A., B. Podolsky and N. Rosen, *Physical Review*, Vol. 47, 1935, p. 777.

[FFS] Feige, U., A. Fiat and A. Shamir, "Zero knowledge proofs of identity", *Proceedings of the 19th Annual ACM Symposium on the Theory of Computing*, 1987, pp. 210-217.

[F] Fortnow, L., "The complexity of perfect zero-knowledge", *Proceedings of the 19th Annual ACM Symposium on the Theory of Computing*, 1987, pp. 204-209.

[GHY] Galil, Z., S. Haber and M. Yung, "A private interactive test of a Boolean predicate and minimum-knowledge public-key cryptosystems", *Proceedings of the 26th Annual IEEE Symposium on the Foundations of Computer Science*, 1985, pp. 360-371.

[GJ] Garey, M. R. and D. S. Johnson, *Computers and Intractability: a Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, 1979.

[G] Gill, J. "Computational complexity of probabilistic Turing machines", *SIAM Journal on Computing*, Vol. 6, no. 4, December 1977, pp. 675-695.

[GMW] Goldreich, O., S. Micali and A. Wigderson, "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*, 1986, pp. 174-187.

[GK] Goldwasser, S. and J. Kilian, "Almost all primes can be quickly certified", *Proceedings of the 18th Annual ACM Symposium on the Theory of Computing*, 1986, pp. 316-329.

[GM] Goldwasser, S. and S. Micali, "Probabilistic encryption", *Journal of Computer and System Sciences*, Vol. 28, no. 2, 1984, pp. 270-299.

[GMR] Goldwasser, S., S. Micali and C. Rackoff, "The knowledge complexity of interactive proof-systems", *Proceedings of the 17th Annual ACM Symposium on the Theory of Computing*, 1985, pp. 291-304.

[GS] Goldwasser, S. and M. Sipser, "Arthur-Merlin games versus interactive proof systems", *Proceedings of the 18th Annual ACM Symposium on the Theory of Computing*, 1986, pp. 59-68.

[M] Mermin, N. D., "Bringing home the atomic world: quantum mysteries for anybody", *American Journal of Physics*, Vol. 49, no. 10, 1981, pp. 940-943.

[Pa] Papadimitriou, C. H., "Games against nature", *Journal of Computer and System Sciences*, Vol. 31, 1985, pp. 288-301.

[Pr] Pratt, V., "Every prime has a succinct certificate", *SIAM Journal on Computing*, Vol. 4, 1975, pp. 214-220.

[R1] Rabin, M. O., "Probabilistic algorithms", in *Algorithms and Their Complexity: Recent Results and New Directions*, J. F. Traub (editor), Academic Press, New York, NY, 1976, pp. 21-39.

[R2] Rabin, M. O., "Digitalized signatures and public-key functions as intractable as factorization", *MIT/LCS/TR-22*, 1979.

[S] Shannon, C. E., "A mathematical theory of communication", *Bell System Technical Journal*, Vol. 27, 1948, pp. 379-423, 623-656.

[SS] Solovay, R. and V. Strassen, "A fast Monte Carlo test for primality", *SIAM Journal on Computing*, Vol. 6, 1977, pp. 84-85.