

# The Application of Claw Free Functions in Cryptography

– Unconditional Protection in Cryptographic Protocols

---

Ivan Bjerre Damgård

DAIMI PB – 269

May 1988

## CONTENTS

1. Introduction	2
2. Basic Definitions and Concepts	3
3. Construction of Function Families	9
4. Collision Free Hash Functions	17
5. Bit Commitment Schemes	27
6. Protocols; Notation and Definitions	33
7. Gradual and Verifiable Release of a Secret	38
8. Multiparty Computations	48
9. Unconditionallly Secure Protocols	75
Table of Abbreviations	78
References	79



## 1. Introduction.

This thesis was written under the guidance of Peter Landrock, Aarhus University. It contains research results, some of which was developed by the author, others were found in joint work with Ernest F. Brickell, David Chaum, Claude Crépeau and Jeroen van de Graaf.

The reader is assumed to have a good knowledge of basic concepts in cryptography, number theory and complexity theory. In particular the reader familiar with minimum knowledge or minimum disclosure protocols will find the sections on protocols much easier.

The main theme in this paper is the application in cryptography of claw free functions, i.e. sets of functions which are easy to compute, but for which it is hard to simultaneously find preimages of an element under different functions. Claw free functions are known to exist under some of the generally accepted but so far unproven assumptions, like the hardness of discrete log or factoring. This paper does not try to prove or disprove the existence of such functions, the emphasis is on their application to various cryptographic constructions. It is worth noting that even if some of the intractability assumptions turn out to be false in a certain sense, some of the applications to protocols would still work!

In the first section we develop a formal framework for describing claw free functions in a complexity theoretic setting, and we describe some constructions of claw free functions, some new and others well known.

We then present some applications, to construction of collision free hash functions, and to unconditionally secure bit commitment schemes. The last concept is used as an essential tool in the protocol constructions that end the paper.

One important aspect of the use of claw free functions in cryptography is that they allow a participant in a protocol to protect his secrets regardless of which computational resources his enemies possess. However, under the standard model for cryptographic protocols, this cannot be achieved in general for more than one participant at a time, as we shall see. It is therefore natural to ask whether more can be achieved under different models of communication. We discuss briefly some recent work by the author, Chaum and Crépeau, where it is shown how to get unconditional protection for *all* participants, assuming that every pair of participants can communicate in secrecy.

The author would like to thank the supervisor, Peter Landrock, and the coauthors of papers, Ernie Brickell, David Chaum, Claude Crépeau and Jeroen van de Graaf for all the hard work and the wonderful inspiration they contributed. Also thanks to Bert den Boer, Joan Boyar, Jørgen Brandt and Jan Hendrik Evertse for many fruitful and stimulating discussions

Finally, special thanks to the external referee, Gilles Brassard, who put a tremendous amount of work into his comments and suggestions for improvements.

## 2. Basic Definitions and Concepts.

This section contains most of the basic definitions needed in this paper. An exception is the definitions related to the security and correctness of a cryptographic protocol. These definitions will be given at the beginning of Section 6. Many of the definitions are quite technical, and the reader unfamiliar with concrete examples of claw free functions and the like is therefore well advised to skip the formal details at a first reading and go directly to Section 3.

### 2.1. Computational Model.

As computational model we will use probabilistic Turing machines (algorithms) throughout this thesis. This is convenient, since communicating Turing machines are natural tools to use in the description of cryptographic protocols, and almost all the main results in this thesis are concerned with construction of such protocols. When in the following, we talk about “the coin flips” of a probabilistic algorithm, this refers to the contents of the random input tape of a probabilistic Turing machine executing the algorithm. Also, when a variable is said to be chosen “uniformly” or “randomly”, this means that the choice is made with the uniform distribution and independently of anything else. It can be argued that in some cases, a non uniform complexity measure would be more suitable in a cryptographic scenario [BIMi]: in real life, the designer of a cryptosystem has to decide on a particular instance of it to use, and only then is it up to an enemy to choose his favorite algorithm for breaking this instance. However, as far as the results in this thesis are concerned, the choice of Turing machines should not be taken too seriously: all results in Sections 3 and 4 could be proved in essentially the same way using for example probabilistic Boolean circuits, and the only change needed would be in the formulation of the intractability assumptions.

### 2.2. Function Families.

#### Definition 2.1

A *function family*  $\mathbf{F}$  is an infinite family of finite sets  $\{\mathbf{I}_m\}_{m=1}^{\infty}$ .  $\mathbf{I}_m$  is called the *set of instances* of size  $m$ . An instance  $S \in \mathbf{I}_m$  is a tuple,

$$S = (f_0, \dots, f_{r_m-1}, U_0, \dots, U_{r_m-1}, V),$$

where for all  $0 \leq i \leq r_m-1$ ,  $U_i$  and  $V$  are finite sets and  $f_i$  is a function:

$$f_i: U_i \rightarrow V.$$

$r_m$  is called the *set size* of  $\mathbf{F}$  and is polynomially bounded as a function of  $m$ . We require that

$$\bigcup_{i=0}^{r_m-1} \text{Im}(f_i) = V.$$

- There is a probabilistic polynomial time algorithm, which on input  $m$  outputs an instance chosen uniformly from  $I_m$ . Also, there is a probabilistic polynomial time algorithm which on input  $S$  and  $i$  selects an element uniformly in  $U_i$ .
- There is a probabilistic polynomial time algorithm which on inputs  $S, i$ , and  $x \in U_i$  computes  $f_i(x)$ . In the following, we will think of the functions as being given by this algorithm, and thus we will identify the symbol  $f_i$  with the algorithm for computing that function  $\square$

For concrete examples fitting this definition, refer to Section 3.

### 2.2.1. Remarks on Choosing Instances.

In Definition 2.1, we have assumed that an instance of a function family can be selected uniformly in probabilistic polynomial time. For the concrete examples we have in mind, the problem that must be solved typically is the following: select uniformly from a given interval, an integer with known factorization. When primality test is assumed to be a primitive, this problem has been solved by Bach [Ba], and thus a theoretical solution is implied by this and the primality test of Adleman and Huang [AdHu]. For the size of numbers we would like to consider, the algorithm from [AdHu] is far from being practical, however, and one would of course like to allow the use of more practical algorithms like Rabin's test, for example, although this introduces a slight deviation from the uniform distribution.

This problem is relevant in the intractability assumptions contained in definitions 2.2 - 2.4 below, and also in all intractability assumptions in Section 3. Basically, these assumptions say that a particular problem cannot be solved in polynomial time for more than a negligible *fraction* of the instances. When instances are not selected uniformly, a more correct statement clearly is: the instances for which the problem can be solved are only *selected* with negligible probability. Incorporating this into the assumptions would mean significantly more complicated definitions and notation, however, and we have therefore chosen the simpler approach of abstracting away the effect of the algorithm actually used for selection of instances, to improve readability of the thesis. This is also justified by the fact that a good approximation to a uniform choice *can* be achieved in practice for most function families, as mentioned above.

From a cryptographic point of view, one should be aware of the fact that, regardless of the choice of formalism, a basic assumption is *always* needed in this context, namely that life is not made easier for an enemy by the fact that a special algorithm is used, which may not produce uniformly chosen instances.

### 2.2.2. Special Function Families.

Definition 2.1 only becomes interesting, if some extra conditions are also satisfied. This leads to the next three definitions.

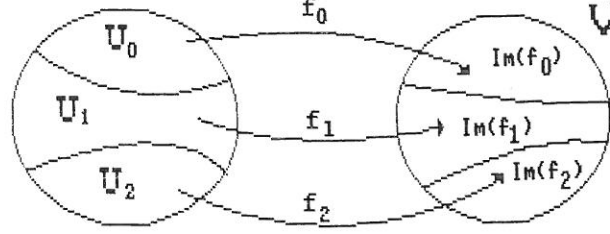


Fig. 1. An instance from a family of probabilistic encryption functions.

### Definition 2.2

A family of probabilistic encryption functions is a function family for which each instance  $S = (f_0, \dots, f_{r_m-1}, U_0, \dots, U_{r_m-1}, V)$  satisfies that

$$\text{Im}(f_i) \cap \text{Im}(f_j) = \emptyset$$

for  $i \neq j$ , but given  $y \in V$  it is a hard problem to decide for which  $j$  it is true that  $y \in \text{Im}(f_j)$ : no probabilistic polynomial time algorithm should be able to do essentially better than guessing at random.

More formally, let  $\Delta$  be a probabilistic polynomial time algorithm, which takes as input a description of  $S \in \mathbb{I}_m$  and a  $y \in V$ , and outputs an integer  $\Delta(S, y)$  with  $0 \leq \Delta(S, y) \leq r_m$ . Let  $p_\Delta(S)$  be the probability that  $\Delta$  “answers correctly”, i.e. that  $y \in \text{Im}(f_{\Delta(S, y)})$ . This probability is taken over all  $y \in V$  and all coinflips of  $\Delta$ . Finally, let  $P$  be any polynomial and let  $\varepsilon(m)$  be the fraction of the instances  $S \in \mathbb{I}_m$  for which

$$p_\Delta(S) > \frac{1}{r_m} + \frac{1}{P(m)}.$$

The requirement is now that for any polynomial  $Q$ ,

$$\varepsilon(m) < \frac{1}{Q(m)},$$

for all sufficiently large  $m$   $\square$

These functions can be used for encryption as follows: given an instance as described above, one can encrypt an integer  $j$  with  $0 \leq j \leq r_m$  by choosing an element  $x$  uniformly from

$U_j$  and computing the encryption as  $f_j(x)$ . Although decryption is unique because the images are disjoint, it is by assumption a hard problem to actually find  $j$  given  $f_j(x)$ .

Probabilistic encryption was introduced by Goldwasser and Micali [GoMi], and the above definition is actually a generalization of the concept of an unapproximable predicate as defined in [GoMi] (they also included a trapdoor property in their definition).

Although probabilistic encryption is an important tool in protocol design, as we shall see, the main emphasis in this thesis is on the concept of clawfree functions:

### Definition 2.3

A *family of claw free functions* is a function family with the property that for any instance  $S = (f_0, \dots, f_{r_m-1}, U_0, \dots, U_{r_m-1}, V)$ , all  $f_i$ 's are  $t$  to 1 mappings for some constant  $t$  and

$$\text{Im}(f_i) = V,$$

for  $i = 0, \dots, r_m-1$ , but given  $S$  it is a hard problem to find "a claw", i.e.  $(x, y, i, j)$ , such that  $x \in U_i, y \in U_j, i \neq j$  and  $f_i(x) = f_j(y)$ .

The formal statement is as follows: Let  $\Delta$  be a probabilistic polynomial time algorithm which takes as input a description of  $S$  and outputs  $i, j$  and two elements  $x_\Delta \in U_i$  and  $y_\Delta \in U_j$ . Let  $p_\Delta(S)$  be the probability that  $f_i(x_\Delta) = f_j(y_\Delta)$  and that  $i \neq j$ . Finally, let  $P$  be any polynomial and let  $\epsilon(m)$  be the fraction of the instances in  $\mathbf{I}_m$  such that

$$p_\Delta(S) > \frac{1}{P(m)}.$$

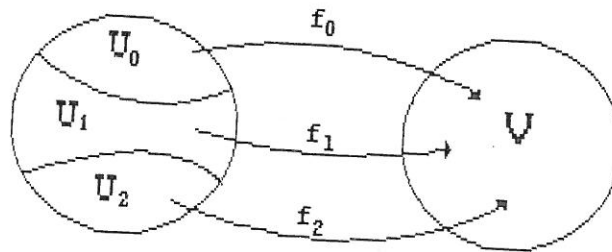


Fig. 2. An instance from a family of claw free functions.

The requirement is now that for any polynomial  $Q$ ,

$$\epsilon(m) < \frac{1}{Q(m)},$$

for all sufficiently large  $m$ . If in particular  $U_0 = U_1 = \dots = V$ , we speak of a family of claw free *permutations*  $\square$

As we shall see, claw free functions can be used in protocols to commit to bits in an unconditionally secure way or to construct collision free hash functions. In earlier work, Goldwasser, Micali and Rivest [GoMiRi] used claw free trapdoor permutations to design the well known GMR-signature scheme which is secure against chosen message attacks. In this thesis, however, we will only consider applications where the trapdoor property is not needed.

Finally, we define what is meant by a one-way function family:

#### Definition 2.4

A *one-way function family* is a function family with  $r_m = 1$  for all  $m$ . Thus any instance  $S \in \mathbf{I}_m$  has the form  $S = (f_0, U_0, V)$ . In this case, we set  $f_0 = f$  and  $U_0 = U$ . We require that inverting more than a negligible fraction of the instances of a given size is a hard problem.

Formally, let  $\Delta$  be a probabilistic polynomial time algorithm which takes as input an element  $f(x) \in V$  and outputs an element  $\Delta(f(x)) \in U$ . Let  $p_\Delta(f)$  be the probability that  $f(x) = f(\Delta(f(x)))$ , where the probability is taken over all  $x \in U$  and all coinflips of  $\Delta$ . Let  $P$  be any polynomial, and let  $\epsilon(m)$  be the fraction of instances  $(f, U, V) \in \mathbf{I}_m$  for which

$$p_\Delta(f) > \frac{1}{P(m)}.$$

We now require that for any polynomial  $Q$ ,

$$\epsilon(m) < \frac{1}{Q(m)},$$

for all sufficiently large  $m$ . If in particular  $U = V$ , we speak of a one way *permutation* family  $\square$

Thus we require that a one way function be hard to invert almost everywhere. As shown by Yao [Kr] the existence of permutations with this property is in fact implied by the existence of permutations which are one way in a much weaker sense. Moreover, the concrete examples we will use in this thesis all involve some group structure. This structure can easily be used to show that if the function involved could be inverted on a non negligible fraction of its images, then it could be inverted everywhere with non negligible probability. This idea was introduced in [GoMi].



### 2.3. Discussion of the Definitions.

In the definitions in the previous section, we have implicitly adopted one of the standard paradigms of complexity theory, namely that anything which can be done in polynomial time is easy, and that all fractions and probabilities that asymptotically vanish faster than any polynomial fraction are negligible. We have chosen to do so for uniformity with other work in the area, and for mathematical convenience. It is, however, extremely important that one is aware of the limitations of this approach.

First of all, any practical application of constructions like this must involve a choice of one specific instance to use. Knowledge of the *asymptotic* behavior of algorithms therefore is of questionable value. In this case the relevant question is whether it is feasible within the given practical framework and current state of the art to solve whichever hard problem your system is based on. The answer to this question may be totally independent of the existence of a polynomial time algorithm for solving the problem! For example, if one claims that RSA is safe to use for moduli of length 512 bits or larger, this is a statement about state of art of factoring of integers of this size, more than it is a statement about the existence of a polynomial time factoring algorithm.

Another consequence of the “polynomial paradigm” is that if you can reduce the solution of a problem  $P_1$  to solving another problem  $P_2$  by a polynomial time reduction, then  $P_2$  is regarded as being at least as hard as  $P_1$ . The problem with this in a cryptographic setting is that nothing has been said about the practical difficulty of the reduction. As an example, consider the case where  $P_1$  is the problem of factoring integers, while  $P_2$  is the problem of breaking some cryptosystem. What does the fact that a polynomial time reduction exists from  $P_1$  to  $P_2$  tell us about the cryptographic strength of a particular instance of the cryptosystem? Perhaps nothing! The only way to use the reduction in a security analysis of a concrete instance is to assume that an enemy can break the cryptosystem in a certain amount of time and then find out, by using the reduction, how fast a factoring algorithm this would yield. Even though the reduction is polynomial, one may end up with a factoring algorithm which for the size of numbers in question is orders of magnitude slower than the best known algorithms.

All this should not be taken to mean that the polynomial model of the world is useless as far as cryptography is concerned, only that this model does not always reflect very accurately the problems that cryptographers encounter in practice. It is therefore important that before use, any construction of a protocol or cryptosystem is evaluated not just in a complexity theoretic model, but also from a more practical point of view, involving state of the art of algorithms in the area, the nature of reductions involved, etc. This also applies, of course, to the constructions given in this thesis. It is therefore up to the reader in each case to reflect on whether for example this particular reduction to factoring is direct enough to convince him or her about the security of the construction!

### 3. Constructions of Function Families.

We begin with two examples of function families that are (hopefully) one way, and upon which many other constructions can be based.

#### The Squaring Family (SQF):

is a function family with  $r_m = 1$  for all  $m$ . To define an instance  $(f, U, V)$  of size  $m$ , select an integer  $n = pq$ , where  $p$  and  $q$  are  $m$ -bit primes and  $p = q = 3 \pmod{4}$ . Here, and in the following constructions, the selection can be done, for example by first selecting  $p$  and  $q$  as strong primes by Gordon's method [Gor], and multiplying to obtain  $n$ . Put

$$U = V = SQ(n) = \{x \in \mathbb{Z}_n^* \mid x = y^2 \text{ for some } y \in \mathbb{Z}_n^*\}.$$

Then define

$$f(x) = x^2 \pmod{n},$$

for  $x \in SQ(n) \square$

It is easy to check that this is in fact a function family: The condition on  $p$  and  $q$  ensures that squaring is a permutation of  $SQ(n)$ , as was first pointed out by Blum, and finding primes congruent to 3 modulo 4 will take about twice the time you need to find an "ordinary" prime. It is well known, and trivial to prove, that inverting these functions is as hard as factoring the modulus. We therefore need the following intractability assumption:

#### The Intractability Assumption on Factoring (IAF):

Let  $H_m = \{n = pq \mid p, q \text{ } m \text{ bit primes, } p = q = 3 \pmod{4}\}$ , and let  $\Delta$  be a probabilistic polynomial time algorithm. Let  $p_\Delta(n)$  be the probability that  $\Delta$  factors  $n$ , taken over the coinflips of  $\Delta$ . For any polynomial  $P$ , let  $\epsilon(m)$  be the fraction of elements  $n \in H_m$  with  $p_\Delta(n) > \frac{1}{P(m)}$ .

Then for any polynomial  $Q$ ,  $\epsilon(m) < \frac{1}{Q(m)}$  for all sufficiently large  $m \square$

As mentioned, it is easy to see that we have the following

#### Lemma 3.1

Under IAF, SQF is a one way permutation family  $\square$

Along exactly the same lines, we can define a one way permutation family, based on the difficulty of discrete log modulo a prime:

#### The Discrete Log Family (DLF):

is a function family with  $r_m = 1$  for all  $m$ . To construct an instance  $(f, U, V)$  of size  $m$ , select



an  $m$ -bit prime  $p$  together with the factorization of  $p-1$ . This can be done by selecting an integer with known factorization [Ba], adding 1 and testing for primality; or (more reasonable in practice) by “building up”  $p-1$  from below by a variation of Gordon’s method [Gor]. Now define

$$U = V = \mathbf{Z}_p^*.$$

Select at random a generator  $g$  of  $\mathbf{Z}_p^*$ , and define

$$f(x) = g^x \bmod p,$$

for  $x \in \mathbf{Z}_p^*$   $\square$

Once again, it is easy to see that this is indeed a function family: primes can easily be constructed with known factorization of  $p-1$  as described above, and with knowledge of this factorization, one can check whether a randomly chosen element in  $\mathbf{Z}_p^*$  is a generator. In this case, the intractability assumption we need for the one way property is directly contained in the definition of a one way function family:

#### The Intractability Assumption for Discrete Log (IDL):

DLF is a one way permutation family as defined in Definition 2.4  $\square$

Note that we have assumed that part of the description of an instance in DLF is the factorization of  $p-1$ . This means that IDL is based on the belief that knowledge of this factorization does not make the discrete log problem easy. This belief is supported by the behaviour of the currently best known discrete log algorithms: they are not able to benefit from knowledge of the factors of  $p-1$ .

Next, we go on to show some examples of function families with larger sizes of function sets. The first example is a well known candidate for a family of probabilistic encryption functions, based on the difficulty of distinguishing quadratic residues from non residues.

#### The Quadratic Residue Family (QRF):

is a function family with  $r_m = 2$  for all  $m$ . To construct an instance  $(f_0, f_1, U_0, U_1, V)$  of size  $m$ , select an integer  $n$  in  $H_m$  (as defined in IAF). Then put

$$U_0 = U_1 = \mathbf{Z}_n^*,$$

and

$$V = \mathbf{Z}_n^{*+1} = \{x \in \mathbf{Z}_n^* \mid (\frac{x}{n}) = 1\}.$$

Finally define

$$f_0(x) = x^2 \bmod n \quad \text{and} \quad f_1(x) = -x^2 \bmod n,$$

for  $x \in \mathbb{Z}_n^*$   $\square$

From Definition 2.2 we get the following intractability assumption:

**The Quadratic Residuosity Assumption (QRA):**

QRF is a family of probabilistic encryption functions as defined in Definition 2.2  $\square$

Normally, QRA is formulated using any integer with two prime factors of about the same length, and not just those that have them both congruent to 3 modulo 4. We have restricted our attention to Blum integers here, mainly for convenience in the protocol constructions to come.

Apart from the restriction to Blum integers, QRA is exactly equivalent to the Quadratic Residuosity Assumption made in [GoMi], where Goldwasser and Micali used it to construct the first probabilistic public key system. Since then a large number of researchers have used it in various protocol constructions. Cohen and Fisher [CoFi] have generalized this using higher power residues, corresponding to function families with  $r_m > 2$ , which allows encryption of more than one bit at once.

Finally we move on to our main subject: examples of claw free function families.

We first present a general construction showing how to make claw free functions based on any one way group homomorphism. Let  $\mathbf{f}$  be a function family with set size 1 for all  $m$ . Thus each instance in  $\mathbf{f}$  is of the form  $(f, G, H)$ , where  $G$  and  $H$  are finite groups such that the group operations and their inverses are computable in polynomial time, and  $f : G \rightarrow H$  is a homomorphism. Based on this, we construct a function family  $\mathbf{F}$  as follows:

Choose any polynomially bounded function  $g : \mathbb{N} \rightarrow \mathbb{N}$ , and put  $r_m$  for  $\mathbf{F}$  equal to  $g(m)$  for all  $m$ . Chose at random an instance  $(f, G, H)$  from  $\mathbf{f}$  of size  $m$  and  $r_m$  elements uniformly from  $H$ ,  $\{a_0, \dots, a_{r_m-1}\}$ . Then put  $U_0 = \dots = U_{r_m-1} = G$  and  $V = H$ . Finally, define the instance of  $\mathbf{F}$ ,  $(f_0, \dots, f_{r_m-1}, U_0, \dots, U_{r_m-1}, V)$  by

$$f_i(x) = a_i \cdot f(x),$$

for  $x \in G$   $\square$

It is now easy to prove the following Theorem:

**Theorem 3.2**

If  $\mathbf{f}$  is a family of one way group homomorphisms, then  $\mathbf{F}$  constructed as above is a family of claw free functions.

**Proof.**

First it is easy to check that  $\mathbf{F}$  is a function family according to Definition 2.1. Secondly since

any instance in  $\mathbf{f}$  is a group homomorphism, it is a  $t$  to 1 mapping for some  $t$  (namely  $t$  equals the order of the kernel of the homomorphism), and therefore so are all maps in  $\mathbf{F}$ . We therefore only need to prove the claw free property. Assume there exists an algorithm  $\Delta$  contradicting the claw freeness of  $\mathbf{F}$ . Let an instance  $f: G \rightarrow H$  chosen uniformly in  $\mathbf{f}$  and a fixed element  $a \in H$  be given. Construct a set  $\{b_0, b_1, \dots, b_{r_m-1}\}$ , where  $b_1, \dots, b_{r_m-1}$  are chosen at random from  $G$ . Now put  $a_i = a^{v_i} f(b_i)$ , where each  $v_i$  is chosen uniformly from  $\{0, 1\}$ . Use the  $a_i$ 's to construct an instance of  $\mathbf{F}$  as above and feed this instance as input to  $\Delta$ . It is easy to see that this procedure leads to a uniformly chosen instance of  $\mathbf{F}$  of size  $m$ , and therefore by assumption on  $\Delta$ , we get a claw with non negligible probability. Now observe that knowledge of a claw amounts to knowing an equation of the form

$$a^{v_i} f(b_i) f(x) = a^{v_j} f(b_j) f(y),$$

for  $i \neq j$  and some  $x, y \in G$ . If  $v_i \neq v_j$ , which happens with probability one half, given that a claw has been produced, then since  $f$  is a homomorphism, the equation immediately gives a preimage of  $a$   $\square$

With essentially the same proof, one can show that one of the  $a_i$ 's can always be chosen equal to 1. This gives a slight improvement in efficiency.

As  $\mathbf{f}$  in this construction, one can use SQF or DLF, and thus we can get claw free sets of permutations assuming that factoring or discrete log is hard. Another candidate for a one way group homomorphism could be RSA, which is a multiplicative homomorphism.

A special case of this construction using SQF was first presented by Brassard and Crépeau in [BrCr]. The use of discrete log to construct claw free maps was first suggested by the author (see [Ch]) in 1986. Later, a variation on this idea was discovered independently by Boyar Krentel and Kurtz [BoKrKu]; and Chaum, Damgård and van de Graaf [ChDaGr], in collaboration with Claude Crépeau. This variation is actually a special case of the construction given here. The generalization to any group homomorphism was found independently by the author and Yung-Impagliazzo [YuIm].

Consider now a practical use of the above construction. The (false!) statement one would like to make about this is now that "if an enemy has found a claw for the set of maps we use, by Theorem 3.2, he must have an algorithm for inverting the one way function involved". The problem is of course that knowledge of a claw for that particular set of maps implies knowledge of preimages of some of the chosen elements. An enemy might know these preimages just by chance, and not as a result of being able to invert the function in general. This problem is not severe in practice, since Theorem 3.2 does show that the problem only occurs with negligible probability, under the appropriate intractability assumption. It does, however, pose the natural question: can one construct function families such that finding claws is *directly* reducible to inverting the function involved? As shown by the next construction, the answer is yes, for any size of claw free sets. More specifically, we construct a function family, for which finding claws is reducible to factoring.

### The Squaring Family II (SQF2)

is a function family constructed as follows: Choose any polynomially bounded function  $g: \mathbb{N} \rightarrow \mathbb{N}$ . We define the function set size for this family to be  $r_m = g(m)$ , for all values of  $m$ . Now choose at random an integer  $n = p_1 p_2 \cdots p_t$ , where all the  $p$ 's are  $m$  bit prime numbers congruent to 3 modulo 4, and where  $t$  is the smallest integer such that  $2^{t-1} \geq r_m$ . The set of integers of this form is denoted  $K_m$ .

For each  $a$  prime to  $n$ , define

$$J(a) = \left( \left( \frac{a}{p_1} \right), \left( \frac{a}{p_2} \right), \dots, \left( \frac{a}{p_t} \right) \right).$$

Clearly,

$$QR(n) = \{a \mid J(a) = (1, 1, \dots, 1)\}.$$

The set of all  $\pm 1$   $t$ -tuples forms a group under pointwise multiplication. Let  $G_t$  denote this group modulo the subgroup generated by  $(-1, -1, \dots, -1)$ . Clearly  $J$  induces a surjective homomorphism  $\phi_n: \mathbb{Z}_n^* \rightarrow G_t$ . Choose a set of  $r_m$  elements in  $\mathbb{Z}_n^*$ ,  $A = \{a_0, \dots, a_{r_m-1}\}$  such that  $|\phi_n(A)| = |A|$ . This is clearly possible by choice of  $t$ .  $A$  is called an *injective set of numbers*, whenever it satisfies this condition. We can now define an instance of size  $m$ ,  $(f_0, \dots, f_{r_m-1}, U_0, \dots, U_{r_m-1}, V)$ , by setting  $U_0 = \dots = U_{r_m-1} = V = QR(n)$ , and defining

$$f_i(x) = (a_i x)^2 \bmod n, \text{ for } x \in QR(n) \text{ and } i = 0, \dots, r_m-1$$

□

To prove that finding claws for SQF2 is as hard as factoring, it turns out to be essential that the  $a_i$ 's (and not just their squares) are made public. It might be argued that this could endanger the factorization of  $n$ , since checking whether a set is injective requires knowledge of the factors, and since the set size grows exponentially with the number of factors. To prove that such release of an injective set is not dangerous, we need the following series of technical, but elementary lemmas:

#### Lemma 3.3.

Let  $G$  be a finite abelian group of exponent 2. Let  $S = \{g_1, \dots, g_s\} \subseteq G$ . Then  $\langle S \rangle$ , the subgroup generated by  $S$ , has order at most  $2^s$ , and equality occurs exactly when no  $g_i$  can be expressed as a product of the others.

**Proof.**

Trivial from the fact that all  $g_i$ 's have order 1 or 2 □

#### Lemma 3.4

Let  $G$  be a finite abelian group of exponent 2,  $|G| = 2^s$ . The probability that a randomly chosen subset  $S$  of  $G$  of cardinality  $s$  generates  $G$  is

$$p_s = \frac{2^s-1}{2^s} \cdot \frac{2^{s-1}-1}{2^{s-1}} \cdot \dots \cdot \frac{2-1}{2}.$$

Moreover,

$$p_s \rightarrow p_\infty = \frac{1}{3} - \frac{1}{3 \cdot 7} + \frac{1}{3 \cdot 7 \cdot 15} - \dots \approx 0.289 \text{ for } s \rightarrow \infty$$

**Proof.**

Let  $S = \{g_1, \dots, g_s\}$ . For all  $1 \leq i \leq s$  we have that

$$\text{Prob}(g_{i+1} \notin \langle g_1, \dots, g_i \rangle) = \frac{2^s - 2^i}{2^s} = \frac{2^{s-i} - 1}{2^{s-i}},$$

provided  $\langle g_1, \dots, g_i \rangle$  has maximal size, by Lemma 3.3. Thus we get

$$\text{Prob}(|\langle g_1, \dots, g_{i+1} \rangle| = 2^{i+1}) = \frac{2^{s-i} - 1}{2^{s-i}} \cdot \text{Prob}(|\langle g_1, \dots, g_i \rangle| = 2^i).$$

This proves the first statement. The second follows from a long series of tedious and rather trivial manipulations with the expression for  $p_s$ . The reader will be spared the details  $\square$

### Lemma 3.5

Consider a probabilistic polynomial time algorithm that on inputs an integer  $n \in K_m$  and an injective set for  $n$ , factors  $n$  with probability  $p(n)$ . For any such algorithm and any  $\varepsilon > 0$  there exists another probabilistic polynomial time algorithm that factors  $n$  with probability  $q(n)$ , without being given the injective set, such that  $|p(n) - q(n)| < \varepsilon$ .

**Proof.**

The idea is to guess an injective set and to feed this and  $n$  to the factoring algorithm we are given. It suffices to consider the case where the set size  $r_m$  equals  $2^{t-1}$  (smaller sets are easier to guess). Choose  $t-1$  numbers at random from  $\mathbf{Z}_n^*$ , call them  $a_1, \dots, a_{t-1}$ . Under  $\phi_n$ , they correspond to  $t-1$  randomly chosen elements in  $G_t$ . By Lemma 3.4, if we form the set

$$A = \{x \in \mathbf{Z}_n^* \mid x = \prod_i a_i^{v_i}, v_i = 0 \text{ or } 1\},$$

then the probability that  $\phi_n(A) = G_t$  (and hence that  $A$  is an injective set) is at least  $p_\infty$ , because the  $p_s$  from Lemma 3.4 is a decreasing function of  $s$ . Since  $r_m$  is at most polynomial in  $m$ , the process of choosing  $A$  can be completed by a polynomial time algorithm. By going through this procedure  $s$  times, we obtain for the probability of success,  $q(n)$ , that

$$p(n) \geq q(n) \geq p(n)(1 - (1 - p_\infty)^s).$$

Since  $p_\infty$  is independent of  $n$ , this proves the lemma  $\square$

We clearly need the following intractability assumption:

**The Intractability Assumption on Factoring II (IAF2):**

Let  $\Delta$  be a probabilistic polynomial time algorithm, and let  $n$  be an integer in  $K_m$ . Let  $p_\Delta(n)$  be the probability that  $\Delta$  factors  $n$ , taken over the coinflips of  $\Delta$ . For any polynomial  $P$ , let  $\epsilon(m)$  the fraction of elements  $n \in K_m$  with  $p_\Delta(n) > \frac{1}{P(m)}$ . Then for any polynomial  $Q$ ,  $\epsilon(m) < \frac{1}{Q(m)}$  for all sufficiently large  $m$   $\square$

**Theorem 3.6**

Under IAF2, SQF2 is a claw free family of permutations.

**Proof.**

Using well known algorithms and the Chinese remainder theorem, the  $p$ 's and the  $a$ 's for each  $n$  can easily be selected. Further, each permutation can be evaluated with two modular multiplications. Thus we certainly have a function family of permutations, and it suffices to prove that finding claws is as hard as factoring. Suppose a claw for the set  $(f_0, \dots, f_{r_m-1}, U_0, \dots, U_{r_m-1}, V)$  has been found. So for some  $i \neq j$  we have:

$$f_i(x) = f_j(y)$$

or

$$(a_i x - a_j y) \cdot (a_i x + a_j y) \equiv 0 \pmod{n}.$$

By choice of the  $a$ 's and basic properties of the Jacobi symbol, this means that  $\text{GCD}(a_i x + a_j y, n)$  will produce a nontrivial factor of  $n$ . By Lemma 3.5, we can now efficiently factor  $n$  if claws can be found efficiently. It is left to the reader to check the formal details in the definitions  $\square$

A special case of this construction with  $r_m = 2$  for all  $m$  was used in [GoMiRi]. Although this family is theoretically interesting for the reasons explained above, it is not the most efficient when one wants to construct large sets of claw free maps. This is because a large set requires more factors and therefore a longer modulus, which implies that each single permutation evaluation takes more time. A modulus with many, shorter prime factors will not be safe enough because of factoring algorithms like Lentra's elliptic curve method whose running time depends mostly on the size of the smallest prime factor.

The final example shows that claw free maps need not be permutations:

**The Jacobi Symbol Family (JSF)**

is a function family with set size 2 for all values of  $m$ . To construct an instance  $(f_0, f_1, U_0, U_1, V)$  of size  $m$ , select at random an integer  $n \in H_m$  (as defined in IAF). Put

$$U_0 = \mathbb{Z}_n^*(+1) \text{ and } U_1 = \mathbb{Z}_n^*(-1) = \{x \in \mathbb{Z}_n^* \mid (\frac{x}{n}) = -1\},$$



$$V = SQ(n).$$

Finally define

$$f_i(x) = x^2 \bmod n,$$

for  $x \in U_i, i = 0, 1 \square$

Since we are dealing with Blum integers, each function in JSF is a 2-1 mapping. Further, knowledge of a claw implies knowledge of different square roots of the same element in  $\mathbf{Z}_n^*$ , which in turn immediately implies knowledge of a factor of  $n$ . It is therefore trivial to prove

**Lemma 3.7**

Under IAF, JSF is a family of claw free functions.

JSS has been used in numerous protocol constructions, for example by Chaum in [Ch].

While the above constructions of claw free maps cover those that are most likely to be used in practice, they do not cover all ideas that have been presented in the literature. For example, Brassard and Crépeau [BrCr] present a construction based on the difficulty of computing graph isomorphisms. However, none of the ideas presented have been able to solve an important open problem with claw free functions: while the existence of probabilistic encryption functions can be proved based on the existence of one way functions only (Yao, [Kr]), the most general construction for claw free maps presented so far, is the one given here based on one way group homomorphisms. Thus a remaining open problem is whether claw free functions can be constructed based only on the existence of one way functions.

## 4. Collision Free Hash Functions.

In this section, we present a construction of hash functions. These functions are collision free in the sense that under some cryptographic assumption, it is provably hard for an enemy to find collisions. Assumptions that would be sufficient are IAF, IAF2, IDL, or the (possibly) more general assumption about the existence of claw free families of permutations.

The ability of a hash function to improve security and speed of a signature scheme is discussed: for example, we can combine the RSA-system with a collision free hash function based on factoring to get a scheme which is more efficient and potentially more secure.

Also, the effect of combining the Goldwasser-Micali-Rivest signature scheme with one of our functions is studied. In the factoring based implementation of the scheme using a  $k$ -bit modulus, the signing process can be speeded up by a factor roughly equal to  $k \cdot O(\log(k))$ , while the signature checking process will be faster by a factor of  $O(\log(k))$ .

The results in this section were presented in [Da].

### 4.1. Motivations for Using Hash Functions.

One of the most fascinating features of public key cryptography is the notion of digital signatures. However, for many of the so far proposed schemes a proof of security does not (yet) exist, or they have been shown to be breakable under sufficiently strong attacks. Moreover, a practical implementation of a signature scheme is often made very difficult by the complexity of the algorithms needed in the system. These problems suggest the use of a hash function, some suitable transformation which is applied to a message before signing it. In particular, we would like to mention the following:

- with a block oriented signature algorithm where messages are longer than a block, it is not safe to sign messages block by block: an enemy could remove blocks from a signed message or insert blocks of his choice into a message before it was signed. Thus, some transformation must be used to make a signature depend on all parts of a message.
- if the message space has some algebraic structure, and the signing algorithm behaves too nicely with respect to this structure, the system can be vulnerable to a chosen message attack (see e.g. [De]). A hash function can be used to destroy this algebraic structure.
- usually, the output of the hash function is much shorter than the input, so if the signature algorithm is slower than the hash function used, a considerable amount of time can be gained in an implementation of the scheme.

The need for hash functions has been realized before ([De], [DP]), and several attempts have been made to construct such functions using e.g. DES or RSA as building blocks. However, none of these suggestions have been proved to be secure, and several of the proposals using DES have been proven insecure ([Wi], [Co]). Other variations of hash functions have also been proposed [WeCa]. But the use of these functions, like pseudo random functions [GGM], require that sender and receiver share a secret key. They are therefore suitable for authentication purposes, but do not fit into the scenario of a public key signature scheme. We



would like to have publicly known hash functions that are easy for all users to compute.

## 4.2. Definition of Collision Free Hash Functions.

One of the most basic demands to a good hash function is that it should be hard to find collisions, i.e. different messages hashing to the same value. We therefore begin this section with a formal description of families of collision free hash functions:

### Definition 4.1

A *Family of Collision Free Hash Functions* is a function family with set size  $r_m = 1$  for all  $m$ . For any value of  $m$ , we choose a finite alphabet  $\Sigma_m$ . An instance of size  $m$  is now of the form  $(h, M, A)$ , where  $h: M \rightarrow A$ ,  $M$  is the set of all finite words over  $\Sigma_m$ , and  $A$  is some finite set (we violate the definition of a function family a little here, by allowing the domain of  $h$  to be an infinite set).

We now require that finding *collisions* is a hard problem, i.e. it is hard to find  $x, y \in M$  such that  $x \neq y$  and  $h(x) = h(y)$ .

More formally, let  $\Delta$  be a probabilistic polynomial time algorithm which takes as input a description of  $h$  and outputs two elements  $x_\Delta, y_\Delta \in M$ . Let  $p_\Delta(h)$  be the probability that  $x_\Delta \neq y_\Delta$  and  $h(x_\Delta) = h(y_\Delta)$ . For any polynomial  $P$ , let  $\epsilon(m)$  be the fraction of instances  $h$  of size  $m$  for which  $p_\Delta(h) > \frac{1}{P(m)}$ . Then for any polynomial  $Q$ ,  $\epsilon(m) < \frac{1}{Q(m)}$  for all sufficiently large  $m$   $\square$

A member  $h$  of a collision free family is also called collision free.

## 4.3. Are Collision Free Functions One Way?

Unfortunately, the most precise answer to this question is "yes and no"! The difficulty lies in the fact that there are two possible definitions of what it means for a function to be one way. One could, as is done in this thesis, start with a uniformly chosen element in the domain of the function, and require that the function can only be inverted with negligible probability on the image of this element. Alternatively, one could start with an element chosen uniformly in the codomain. All one way functions (other than hash functions) considered here are  $t$ -1 mappings for some constant  $t$ . For functions like that the two definitions are trivially equivalent, since they produce a uniformly distributed variable when applied to a uniformly distributed one. This is not necessarily the case for a collision free function, however, and therefore we can only prove one of the one way properties for collision free functions:

### Lemma 4.1

Let  $h: M \rightarrow A$  be an arbitrary instance in a collision free family. Consider now the restriction of  $h$  to any finite subset  $W_m$  satisfying  $|W_m| > (1 + \frac{1}{R(m)})|A|$ , where  $R$  is an arbitrary

polynomial. Any function family resulting from this restriction is one way as defined in Definition 2.4.

**Proof.**

Assume the lemma is false, and let  $\Delta$  be an algorithm contradicting the one way property. Let  $h$  be one of the instances of size  $m$  for which  $\Delta$  can find preimages with probability larger than a polynomial fraction. Choose  $x \in W_m$  at random and compute  $h(x)$ . By assumption, we can efficiently find  $x'$  such that  $h(x)=h(x')$ . By assumption on the size of  $A$  and  $W_m$ , the probability that  $x \neq x'$  in which case this procedure leads to a collision for  $h$ , is at least a polynomial fraction  $\square$

It turns out that in connection with chosen message attacks on signature schemes, it is not the one way property proved in the lemma which is interesting. It is therefore important to test any construction of hash functions both for collision freeness and for the one way property not implied by it.

#### 4.4. Construction of Collision Free Hash Functions.

Assuming the existence of claw free permutations, we can construct collision free hash functions. First some notation:

Let an alphabet  $\Sigma$  with cardinality  $t$  and a finite word  $w$  over  $\Sigma$  be given. We now let  $[w]$  denote a prefix free encoding of  $w$  over  $\Sigma$ . The choice of a particular encoding is not important to the results given here, except for the fact that it is possible to encode efficiently so that the length of  $[w]$  is a linear function of the length of  $w$ . In binary, for example, 1 could be encoded as 11, 0 as 00, and all encodings could be terminated with 01. This is important because a short encoding will make the mechanisms shown here more efficient. In fact, theoretical results show that the encoding can be chosen such that the length of  $w$  is almost equal to the length of  $[w]$ . In this case, however, the encoding process itself might become inefficient. Further details on prefix free encodings can be found in [BePe].

Now, if  $\{f_0, \dots, f_{t-1}\}$  is a set of permutations, all with domain  $D$ , we define

$$f_{[w]}(I) = f_{w_1}(f_{w_2}(\dots f_{w_s}(I) \dots)),$$

where  $I \in D$ ,  $[w] = w_1 w_2 \dots w_s$ , and the letters in  $\Sigma$  are denoted by the numbers  $0, \dots, t-1$ . A similar construction (for a different purpose) is used in [GoMiRi] with  $t=2$ .

#### Theorem 4.2

Construct a function family  $F$  as follows:

Let  $P$  be a family of claw free permutations. For each value of the security parameter  $m$ , we let  $\Sigma_m$  be the alphabet of cardinality  $r_m$  given by  $\Sigma_m = \{0, 1, \dots, r_m-1\}$ , where  $r_m$  is the set size for  $P$ . As in Definition 4.1, let  $M_m$  be the set of all finite words over  $\Sigma_m$ . Each instance in  $P$  has the form  $(f_0, \dots, f_{r_m-1}, U_0, \dots, U_{r_m-1}, V)$ . Since we are dealing with

permutations, all  $U_i$ 's and  $V$  are equal to one finite set  $A$ . For each such instance and each  $I \in A$ , we define an instance  $(h, M, A)$  of  $F$  of size  $m$  by:

$$h(w) = f_{[w]}(I),$$

for all  $w \in M$ .

Then  $F$  is a family of collision free hash functions.

**Proof**

It is sufficient to show how to compute from a collision for any  $h \in F$  a claw for the underlying set of permutations. Suppose therefore that we have  $w \neq w'$ , such that

$$f_{[w]}(I) = f_{[w']}(I),$$

where  $[w]$  and  $[w']$  have lengths  $s$  and  $s'$ , respectively. Note that since  $w$  and  $w'$  are assumed to be produced by a polynomial time algorithm, both  $s$  and  $s'$  can be at most polynomial in  $m$ . If  $w_1 \neq w'_1$ , we have a claw for the set  $S$ . If  $w_1 = w'_1$ , the fact that the  $f$ 's are injective implies that

$$f_{w_2}(\cdots f_{w_s}(I) \cdots) = f_{w'_2}(\cdots f_{w'_s}(I) \cdots).$$

The same argument now applies again, and since the encoding used is prefix free, this process must stop with the creation of a claw  $\square$

If we let  $T$  denote the time needed to evaluate one of the permutations used in the construction, it is clear that the time needed to compute  $h$  on a message of length  $L$  is  $O(TL)$ .

Any of the constructions of claw free permutations from the previous section could be used as  $P$  in the theorem. But it is clear that the permutation family resulting from using Theorem 3.2 on SQF is by far the most efficient: Recall that in this scheme, a permutation is evaluated by squaring the input and multiplying by a fixed coefficient chosen for that permutation. Thus each permutation evaluation only costs two modular multiplications, and the set size (which is equal to the alphabet size) can be enlarged without making the evaluation of permutations slower. This can have dramatic effects on efficiency: If one is willing to store (or generate pseudorandomly) a set of 1024 fixed coefficients, the resulting hash function will be 10 times faster than a system using pairs of permutations: we can regard 10-bit chunks of the input as members of an alphabet with 1024 elements, and in this way hash 10 bits in one operation. In general, an alphabet of size  $2^s$  can cover all  $s$ -bit values, and therefore the effect is not quite so dramatic asymptotically. Since we can only store a set of polynomial size, the factor of improvement we get over a system using pairs is  $O(\log_2(m))$ , where  $m$  is the bit length of the modulus used.

As mentioned in the previous section, we must also test our construction for one wayness: given an element chosen uniformly in the image of the function, is it possible to find any preimage?

For the hash functions based on SQF, it turns out that inverting the hash function is as hard as computing discrete log: as a simple example, suppose we base the hash function on the pair of claw free permutations:

$$f_0(x) = x^2 \text{ and } f_1(x) = ax^2,$$

where the computations are modulo some  $n \in H_m$  and  $a \in SQ(n)$ . It is now easy to check that

$$h(w) = f_{[w]}(I) = a^{[w]} \cdot I^{2^{l([w])}},$$

where  $I$  is the randomly chosen starting point mentioned in Theorem 4.2, and  $l([w])$  is the length in bits of  $[w]$ . So when presented with an element in  $Im(h)$ , even if an enemy could (magically) guess the length of a preimage, he would still have to solve a discrete log problem base  $a$  in order to invert  $h$ . Moreover, the set of discrete log problems that could arise in this way certainly constitute a nonnegligible fraction of all possible discrete log problems base  $a$ .

The question becomes much more difficult when we consider hash functions based on claw free permutations in general. It is possible, however, to give a heuristic argument: Consider the case where we base the hash function on a pair  $(f_0, f_1)$  of claw free permutations. Let  $D$  be the common domain of  $f_0$  and  $f_1$ . As an approximation to the actual constructions, assume that  $f_0$  and  $f_1$  are randomly chosen permutations of  $D$ . Since an image under the hash function is computed by applying the  $f$ 's in some sequence to the fixed starting point  $I$ , we can at least say that  $h$  is surjective if the group generated by  $f_0$  and  $f_1$  is transitive on  $D$ . In [Di], Dixon proves that with probability almost 1, this group will not only be transitive, but will in fact be the alternating group or the full symmetric group of degree  $|D|$ . This strongly suggests that the images of  $h$  will be "evenly distributed" over  $D$ , thus implying that the two flavours of one way property are equivalent for these functions. Both of them will therefore be implied by collision freeness.

#### 4.5. Combining a Signature Scheme and a Hash Function.

The model we shall use is the following: we have a signature scheme with

- message space  $A$
- signature space  $B$
- and a signature algorithm  $\Delta$ ,

which for any message  $a \in A$  produces as output a signature  $\Delta(S, a) \in B$ , using the secret key  $S$ .

$\Delta$  could be probabilistic, and its output could depend on the number of messages previously signed, or how they were signed in which case the signature scheme is called non-memoryless. Since memorylessness is not relevant to the results proved below, the dependence on random inputs or previously signed messages has been omitted from the notation. Finally we have

- a verification predicate  $\Gamma$ ,

which for any message  $a$  and signature  $s$  produces as output a boolean value  $\Gamma(P, a, s)$  using the public key  $P$ .  $\Gamma(P, a, s) = \text{TRUE}$  if and only if  $s$  is a valid signature for  $a$  using the secret key matching  $P$ .

This scheme is referred to as *the original scheme*. A full model of a signature scheme would also have to include an algorithm that generates matching pairs of public and secret keys, but this will not be important in this context.

As usual, a number of things such as the length of the keys, the running time of  $\Delta$  and  $\Gamma$ , the overall security of the scheme depend on the value of a security parameter  $m$ .

We now combine this scheme with a hash function  $h: M \rightarrow A$ . A message is signed by computing  $\Delta(S, h(m))$  and signatures are checked by computing  $\Gamma(P, h(m), \Delta(S, h(m)))$ . This scheme will be called *the combined scheme*.

When combining with a collision free hash function, we always assume for simplicity that the signature scheme and the hash function have the same value of security parameter (i.e. are instances of the same size).

Following [GoMiRi], a signature scheme is called *existentially forgeable* if under some attack, an enemy can forge the signature of at least one message. This message cannot necessarily be chosen by the enemy.

A *chosen message attack* is an attack where the enemy can choose messages to be signed by the legitimate signer before trying to forge a signature. If he is also allowed to choose such messages *during* the process of forgery, we speak of an *adaptive chosen message attack*.

It is of course essential to be able to compare the security of the two schemes. A first result in this direction is:

### Theorem 4.3

Suppose that the combined scheme is at least existentially forgeable under a chosen message attack, and that the hash function used is collision free. Then the original scheme can be existentially forged under a chosen message attack.

#### Proof.

Assume for contradiction that we have an algorithm  $\Phi$  that forges at least one signature in the combined scheme after receiving signatures of messages of its choice. Assume also that we can mount a chosen message attack on the original scheme. While running  $\Phi$ , we will be asked for signatures of messages in some subset  $N$  of  $M$ . They are easily found by computing the set  $h(N)$  and using the chosen message attack to obtain  $\Delta(S, h(N))$ . With nonnegligible probability,  $\Phi$  will produce as output a message  $m \notin N$  and a valid signature for  $m$ . If  $h(m) \in h(N)$  we have found a collision for  $h$ , but by assumption on  $h$  this can only occur with negligible probability. We may therefore assume that  $h(m) \notin h(N)$  which means that we have forged a new signature in the original scheme  $\square$



Note that the proof goes through in the same way if the attack is adaptive. Therefore “adaptive chosen message attack” could substituted for “chosen message attack” in the theorem.

This result has two useful implications:

- If the original scheme is not existentially forgeable, then neither is the combined scheme.
- If the combined scheme has some degree of weakness, then the original scheme was already existentially forgeable.

The last fact says that in some sense, the use of the hash function does not introduce any “completely new” weaknesses. This may make a security analysis of the combined scheme easier, but unfortunately it does not rule out the possibility that the combined scheme is much weaker than the original scheme! This will be illustrated by an example later.

The next result, however, shows that at least a large class of attacks directed against the original scheme are prevented by the use of a hash function with the right properties:

#### Proposition 4.4

Suppose that the hash function used is one way in the sense that it is hard to invert on a randomly chosen element in  $A$  (c.f. Section 4.2). Consider an algorithm for forging signatures which is directed against *the original scheme*, i.e. the hash function used is not part of the input to the algorithm. No such algorithm which satisfies i) or ii) below will be of any use in attacking the combined scheme.

- i) The algorithm only works under a chosen message attack, and during the attack it will ask for signatures of messages uniformly distributed in  $A$ .
- ii) The algorithm can only forge signatures for a small (i.e. polynomial) number of messages, which are non-chosen and uniformly distributed over the message space (where the distribution is taken over all choices of messages to sign made by the signer and all choices made by the enemy).

#### Proof

i): by the one way property, even a chosen message attack on the combined scheme does not allow a chosen message attack on the original scheme: any procedure allowing this would in fact be an inversion algorithm for  $h$ . ii): Call the set of messages for which the algorithm has forged signatures  $X$ . Since  $X$  is in fact a random variable, then to use these signatures, the enemy must have an algorithm that on input any set of messages  $Y$  with  $|Y| = |X|$  efficiently produces an element  $m \in M$  with  $h(m) \in Y$ . But this would enable him to invert  $h$  on a randomly chosen element with non negligible probability  $\square$

Recall that the hash functions based on SQF provably have the one way property required in the proposition, assuming discrete log is hard, and that it is at least reasonable to assume that any function based on claw free permutations have the property.

Assumptions i) and ii) above seem to be reasonable models of the multiplicative attacks on RSA mentioned in [De]. Thus our hash functions would prevent these. But let us stress once again that the proposition does not talk about attacks directed against the *combined* scheme. Therefore an analysis of this combination is needed in each concrete case.

As an example of these problems, consider a combined scheme with RSA as the original scheme, and a hash function based on SQF as described in Section 4.3.

If the modulus used for the hash function is chosen *independently* of the RSA-modulus, then the following assumption seems reasonable:

If there exists an algorithm that forges RSA signatures under condition 1) below, then there exists an equally efficient algorithm that forges signatures under condition 2).

- 1) All messages signed are of the form  $h(m)$ , and  $h$  and  $m$  are known to the forgery algorithm.
- 2) All messages are chosen at random from  $\text{Im}(h)$ .

In other words, as far as forging RSA-signatures is concerned, the hash function might as well be a random function. Unfortunately, no satisfactory definition is known of what it means in general that a publicly known function can “simulate” a random function. It is therefore very hard to approach a proof of this assumption. But if it is indeed true, then by Proposition 4.4 the only way to break the combined scheme is to totally break RSA under a non chosen message attack (i.e. forge the signature of any message with nonnegligible probability). It therefore seems that this combination is much more secure than plain RSA.

A word of warning, however: if the two moduli used are not independent, then the above assumption can be seriously violated. For example, if for convenience we use the same modulus for the hash function as for the RSA, then by an attack due to Bert den Boer [Bo], the combined scheme can in fact be broken. The attack relies heavily on the fact that the multiplicative structure used in computing the hash function is exactly the same as the one used in computing signatures: As explained in Section 4.4, a simple formula exists for the output of the hash function, involving the modulus used for computing that output. If this modulus is also used for signing, then the formula can be used to find multiplicative relations between hashed values. This, and the homomorphism property of RSA, can then be used to break the system. Clearly, this attack does not generalize to the case where the moduli are independently chosen.

It is worth noting that in addition to being more secure, the combination with RSA has a number of practical advantages:

- Since modular arithmetic is used for both the hash function and RSA, the combination can be implemented without using more hardware than is needed for basic RSA.
- The combined scheme is more efficient than plain RSA on long messages. As mentioned in Section 4.2, if one is willing to store a set of for example 512 constant coefficients, then the combined scheme will be about 9 times faster than the original one.

- Signatures have length 1 RSA block, independent of the message length.

#### 4.6. Speeding up the Goldwasser Micali Rivest Signature Scheme.

In [GoMiRi], the Goldwasser-Micali-Rivest (GMR) signature scheme is introduced and is proven to be not existentially forgeable, even under an adaptive chosen message attack. The basic building blocks in the scheme are two pairs of claw free trapdoor permutations,  $(f_0, f_1)$  and  $(g_0, g_1)$ . The trapdoor property means that a certain piece of information known as the trapdoor goes with each pair, and knowledge of the trapdoor allows easy inversion of the permutations. It must be possible, however, to release enough information to allow evaluation of the permutations without compromising the trapdoor.

To sign a message  $a$ , the following is done:

- 1) A random value  $R$  is chosen.
- 2) A value  $\alpha$  is computed using the trapdoor information for the  $g$ -permutations such that  $g_{[a]}(\alpha) = R$ .
- 3)  $R$  is authenticated using the  $f$ -permutations.

The signature for  $a$  now consists of  $\alpha$ ,  $R$ , and the authentication for  $R$ . It is pointed out that the scheme is most practical when messages have length much greater than the value of security parameter used, since in this case signatures are short compared to the messages. Clearly, if  $\alpha$  is computed in a straightforward way by inverting the  $g$ -permutations involved one by one, then the time needed to compute a signature for a message of length  $L$  is  $O(LT_{inv})$ , where  $T_{inv}$  is the time needed to invert one permutation. Moreover, if (as proposed in [GoMiRi]) we use the factoring based construction of claw free pairs of permutations, then  $T_{inv}$  is greater than the time for one permutation evaluation by a factor of  $m$ , where  $m$  is the length of the modulus used.

Using the constructions from Section 2, we can improve the efficiency of this scheme substantially:

- 1) It is trivial to check that the GMR scheme will work equally well with any size of sets of claw free permutations. Therefore we can use the construction from Section 4.2 in place of both the  $f$ - and the  $g$ -permutations. By the remarks in Section 4.2, this will speed up both the signing and the signature checking process by a factor of  $O(\log_2(m))$ .
- 2) By combining with a collision free hash function based on factoring, we can avoid most of the permutation inversions. It is easy to check that this will speed up the signing process further by a factor of about  $m$  for long messages. Moreover, by Theorem 4.2 and 4.3, the combined scheme is as secure as the original one.

Other methods for speeding up the GMR scheme have been proposed, both by Goldreich [Go] and Goldwasser, Micali and Rivest in the full version of their paper [GoMiRi2]. They both apply only to the signing process and achieve a slightly smaller improvement than our method (the factor gained is  $m$ ). They are also fundamentally different: the method from [Go]



only applies to the factoring based implementation of claw free permutations, while our method is potentially useful with any implementation. The method from [GoMiRi2] will not allow the use of non trapdoor permutations, while the construction of a hash function only uses the claw free property.

## 5. Bit Commitment Schemes.

This section focuses on the application of function families in cryptographic protocols. The application we have in mind is the construction of *bit commitment schemes*. Basically, a bit commitment scheme is a tool allowing protocol participant  $A$  to *commit* to a bit by releasing some information related to it such that:

- she cannot later change her mind about the bit.
- No one else can find the value of the bit unless  $A$  allows this (opens the commitment).

We have the following definition:

### Definition 5.1

A *Bit Commitment Scheme* is a function family with set size 2 for all instances.

To use a bit commitment scheme, an instance  $(f_0, f_1, U_0, U_1, V)$  will be selected and made public, where

$$f_0: U_0 \rightarrow V \text{ and } f_1: U_1 \rightarrow V.$$

To commit to  $b \in \{0,1\}$ ,  $A$  will select  $r$  in  $U_b$  and compute her commitment as

$$BC_A(b, r) = f_b(r).$$

Certain properties must be satisfied in order for a commitment scheme to be of any use in protocols:

*The Hiding Property:*

Given a commitment to  $b$ ,  $BC_A(b, r)$ , it is hard to find  $b$ : it should be at least computationally infeasible to do better than guessing at random.

*The Unforgeability Property:*

It is a hard problem to find  $r \in U_b$  and  $r' \in U_{b \oplus 1}$  such that  $BC_A(b, r) = BC_A(b \oplus 1, r')$ .

*Opening a Commitment.*

Given  $BC_A(b, r)$ ,  $A$  can convince anyone that the commitment really represents  $b$  (and not  $b \oplus 1$ ). Typically, she will just reveal  $r$   $\square$

Thus, the concept of a commitment scheme is very similar to that of a “blob”, an even more general and abstract concept invented by Bennet and Brassard, and described in [BrChCr].

Apart from these three properties, a fourth one is implicit in the fact that the functions used are made public. This means that anyone, not just the creator of the instance, can create blobs, and it will be impossible to tell from a blob who created it. This will be important to the simulation proofs given later.

The three basic properties are trivially satisfied by any bit commitment scheme constructed from a family of probabilistic encryption functions or a family of claw free functions:

**Lemma 5.1**

Any bit commitment scheme based on a family of probabilistic encryption functions satisfies the Hiding property computationally and satisfies the Unforgeability property unconditionally.

**Proof.**

For a family of probabilistic encryption functions, the Hiding property is equivalent to the intractability assumption contained in Definition 2.2. For the Unforgeability property, note that the  $r'$  mentioned above does not exist in this case because the images of  $f_0$  and  $f_1$  are disjoint  $\square$

**Lemma 5.2**

Any bit commitment scheme based on a family of claw free functions satisfies the Hiding property unconditionally, and satisfies the Unforgeability property computationally.

**Proof.**

For the Hiding property, note that commitments to 1's and 0's have the same distribution (uniform in  $V$ ), because both  $f_0$  and  $f_1$  are  $t$ -1 mappings for some constant  $t$ , and therefore commitments give away no Shannon information about the bits they contain. The Unforgeability property is exactly equivalent to the intractability assumption contained in Definition 2.3  $\square$

Two other properties will sometimes be desirable:

*Comparability:*

Given two commitments,  $BC_A(b, r)$  and  $BC_A(b', r')$ ,  $A$  can convince anyone about the value of  $b \oplus b'$  while revealing *nothing* more about  $b$  and  $b'$ .

For some commitment schemes, it is only possible to prove efficiently that  $b = b'$  (if this is so!). With some loss of efficiency, one can then also prove inequality using a subprotocol.

*The Blinding Property:*

Given a commitment  $BC_A(b, r)$ , another participant  $B$  can choose  $b' \in \{0, 1\}$  and compute from this a *blinded* commitment  $BC_A(b \oplus b', r')$ , where  $r'$  is uniformly distributed in  $U_{b \oplus b'}$ . Moreover, given these two commitments,  $B$  can convince anyone about the value of  $b'$ . From the two commitments only, however, no participant other than  $A$  or  $B$  can guess the value of  $b$ ,  $b'$  and  $b \oplus b'$  essentially better than at random.

We also require that given two commitments  $BC_A(b \oplus b', r')$  and  $BC_A(b \oplus b'', r'')$  which have been computed as above by  $B$ , he can convince anyone about the value of  $(b \oplus b') \oplus (b \oplus b'') = b' \oplus b''$ .

Finally, it must be possible for  $A$  to open commitments which have been blinded in this way, just as if she had computed them herself.

From a theoretical point of view, the comparability property does not represent a demanding assumption, since the required proof could always be produced in a minimum

knowledge fashion by using standard reductions to 3COL or SAT ([GoMiWi2], [Ch], [BrCr]). We will only be concerned, however, with bit commitment schemes where special properties allow the proofs to be produced directly and efficiently. By contrast, for some commitment schemes, blinding cannot be achieved, but requires special properties, as we shall see.

It will be of interest later to see also how bits hidden using *different* bit commitment schemes can be compared (this protocol was found by Jeroen van de Graaf): Let  $BC\ 1$  and  $BC\ 2$  be two different instances of bit commitment schemes, both satisfying the comparability property. Then given commitments  $BC\ 1_A(b_1, r_1)$ ,  $BC\ 2_A(b_2, r_2)$ , the protocol below can be used by  $A$  to convince  $B$  that  $b_1 \oplus b_2 = b$  for a given value of  $b$ .

#### PROTOCOL COMPARE COMMITMENTS.

- 1)  $A$  computes and sends to  $B$  a pair of commitments  $BC\ 1_A(b', r_1')$  and  $BC\ 2_A(b', r_2')$ , where  $b'$  is randomly chosen.
- 2)  $B$  flips a coin and requests either
  - (a) opening by  $A$  of both commitments sent in Step 1), or
  - (b) to be convinced by  $A$  about  $b_1 \oplus b'$  and  $b_2 \oplus b'$ .
- 3) On request (a),  $A$  opens both commitments from Step 1).  
On request (b),  $A$  uses comparability of  $BC\ 1$  and  $BC\ 2$  to convince  $B$  about  $b_1 \oplus b'$  and  $b_2 \oplus b'$ .
- 4) On request (a),  $B$  checks that the two commitments did indeed contain the same  $b'$ .  
On request (b),  $B$  checks that  $(b_1 \oplus b') \oplus (b_2 \oplus b') = b$ .

Steps 1) - 4) are repeated  $m$  times.

Using the formal machinery presented in the next section, it is easy to prove that this protocol convinces  $B$  with very large probability that  $b_1 \oplus b_2 = b$ , and that it is minimum knowledge, i.e. does not release anything but this fact.

### 5.1. Examples of Commitment Schemes.

From the preceding sections it is clear that we can get commitment schemes based on all the usual intractability assumptions, IAF, IDL and QRA, by using the construction above on any of the function families we have seen in Section 3. The functionality you obtain from the resulting commitment schemes can be very different, however. Let us therefore have a look at some concrete examples:

Consider first using QRF to produce a bit commitment scheme. This scheme will be called QRS. In this scheme,  $A$  will choose a Blum integer  $n$ , and prove to all other participants in minimum knowledge that her choice was correct. This can be done for example with the protocol from [GrPe]. All commitments will be numbers of Jacobi symbol 1, and  $A$  will publish modular squares to commit to 0's and non squares to commit to 1's. This is an

example of bit commitments constructed by probabilistic encryption, so by the above the Hiding property is satisfied relative to QRA, while the Unforgeability property is unconditionally satisfied. To prove that a commitment  $BC_A(b, r)$  represents  $b$ ,  $A$  will make public a square root modulo  $n$  of  $(-1)^b BC_A(b, r)$ . Note that this does not work if  $n$  is not a Blum integer, which is part of the reason why it is necessary for  $A$  to prove that her choice of  $n$  is correct initially. Comparability is also easy: to convince everybody about the value of  $b \oplus b'$  from commitments to  $b$  and  $b'$ ,  $A$  makes public a square root of

$$(-1)^{b \oplus b'} BC_A(b, r) (BC_A(b', r'))^{-1}.$$

Finally, QRS also satisfies the Blinding property: given  $BC_A(b, r)$ ,  $B$  will choose  $b' \in \{0, 1\}$  and  $s \in \mathbb{Z}_n^*$  and compute the blinded commitment as

$$(-1)^{b'} s^2 BC_A(b, r) = BC_A(b \oplus b', rs).$$

$B$  can show the value of  $b'$  by showing a square root of

$$(-1)^{b'} BC_A(b \oplus b', rs) (BC_A(b, r))^{-1},$$

which equals  $s$ .  $s$  is called *the blinding factor*. Also, given two commitments blinded as above,  $B$  can show a square root of

$$(-1)^{b' \oplus b''} BC_A(b \oplus b', r') (BC_A(b \oplus b'', r''))^{-1},$$

to convince everyone about the value of  $b' \oplus b''$ . Clearly,  $A$  can use her knowledge of the prime factors of  $n$  to open any commitment by computing the square roots needed. But it is important to note that it will only be safe for her to do so, if she will always be opening blinded forms of commitments chosen by *herself*. We will see later how this can be ensured by construction of a protocol. Moreover, we have not yet proved that the comparability proof above does not leak anything useful about the hidden bits. A proof of this will be implicit in the security proofs of the protocols to come later.

A different example of bit commitments is the Jacobi Symbol Scheme (JSS) which is based on the family of claw free functions JSF. With this scheme, we cannot allow  $A$  to choose the modulus involved, since knowledge of the factors allows easy computation of claws, which would violate the unforgeability property. So in this case some other participant,  $B$ , chooses a Blum integer  $n$  and proves to  $A$  in minimum knowledge that the choice was correct. All commitments will now be in  $SQ(n)$ , and for each commitment,  $A$  knows a square root of it modulo  $n$ . The Jacobi symbol of this root determines the bit she is committed to. As all commitment schemes based on claw free functions, JSS hides the bits unconditionally, while the unforgeability property is only satisfied relative to some cryptographic assumption - IAF in this case. In other words, if  $A$  could factor  $n$ , she could lie about the bits she commits to. JSS also satisfies the comparability property: Given commitments  $BC_A(b, r)$  and  $BC_A(b', r')$ ,  $A$  can make public a square root of

$$BC_A(b, r) (BC_A(b', r'))^{-1}$$

of Jacobi symbol  $(-1)^{b \oplus b'}$  to convince everyone about the value of  $b \oplus b'$ . Note that this is still information theoretically secure: after having seen a proof that  $b \oplus b' = 1$ , say, the two possibilities  $(b, b') = (1, 0)$  and  $(b, b') = (0, 1)$  remain equally likely, even to someone who knows the factors of  $n$ .

Things get a little more complicated when JSS is to be used in a multiparty protocol. We must make sure that  $A$  does not break the unforgeability property by conspiring with somebody who knows the factorization of  $n$ . One way out is to let every participant except  $A$  supply a different modulus, and when committing to a bit,  $A$  must make public a set of commitments to this bit, one for each modulus supplied. Another difficulty with JSS is that all known proofs that  $B$  could use to convince  $A$  that  $n$  really is a Blum integer leave a very small probability that  $B$  is lying (in fact exponentially small in the amount of work done in the protocol). It is easy to see that the Hiding condition is seriously violated if  $n$  is not a Blum integer. So even if  $B$  is required to reveal the factorization of  $n$  after the main protocol is finished, we still have a situation where  $A$  can never be quite sure that her secrets have not been betrayed until after it happened! In the language of information theory, we can say that a protocol using JSS leaks an exponentially small amount of information on the average. A similar problem occurs with the commitment scheme used in [BrCr] (this scheme is essentially the one obtained by applying Theorem 3.2 to SQF - it is extremely efficient in practice, despite the theoretical drawback).

Fortunately, all these difficulties can be removed by using the discrete log problem instead: construct a bit commitment scheme from the family of claw free functions you get by using Theorem 3.2 on DLF. This will be called the Discrete Log Scheme (DLS). In this scheme all participants can cooperate in choosing a large prime  $p$ , a generator  $g$  of  $\mathbb{Z}_p^*$ , and an element  $a \in \mathbb{Z}_p^*$ . No secret trapdoor is present here, so all participants including  $A$  can check for themselves that the choices are correct. All commitments will be uniformly distributed in  $\mathbb{Z}_p^*$ , and to open a commitment  $A$  has to make public the discrete log base  $g$  of

$$a^{-b} BC_A(b, r),$$

namely  $r$ . The comparability property also holds: given commitments to  $b$  and  $b'$ ,  $A$  can prove that  $b = b'$  by showing the discrete log of

$$BC_A(b, r)(BC_A(b', r))^{-1}$$

and to prove that  $b \neq b'$ , she shows the discrete log of

$$a^{-1} BC_A(b, r) BC_A(b', r).$$

It is easy to see that  $A$  cannot make a false claim about  $b \oplus b'$  unless she can find the discrete log of  $a$ , and that the unconditional hiding of bits is preserved by the use of these proofs.

## 5.2. Claw Free Functions versus Probabilistic Encryption.

It is worth noting some very fundamental differences that exist between protocols using claw free functions for bit commitments and protocols using probabilistic encryption.



Consider for example a protocol where  $A$  uses QRS to protect her secrets. This means that  $A$  is in fact giving away full Shannon information about her secrets, because every commitment uniquely determines the bit it contains. The security of the protocol therefore only rests on the hope that this information will not be easy to extract from the protocol conversation. On the other hand,  $A$  cannot lie about her secrets, even if she has infinite computing power. The important point here is that if  $A$ 's modulus is factored at any time after execution of the protocol, all her secrets will be revealed. By contrast, if  $A$  uses for example JSS, the secrets will be unconditionally protected. And even though  $A$  could lie if she could factor, it is important to note that she has to be able to do so *before* the protocol is finished, after this point the factorization is completely useless to her.

The bottom line is that even if some of the intractability assumptions turn out to be false, if even a large modulus could be factored in a few days for example, all the protocols using claw free functions might still be useful, while protocols using QRS would become hopelessly insecure. A detailed discussion of these and related problems can be found in [BrCr].

## 6. Protocols; Notation and Definitions.

This section contains the formal setup we will use in describing protocols. Also, the basic definitions of security and correctness are given. These definitions are generalizations of corresponding definitions for zero-knowledge proof systems as defined in [GoMiRa].

We think of a protocol as occurring among a set of communicating probabilistic polynomial time Turing machines  $\{P_1, \dots, P_n\}$  called the *participants*. The protocol is described as a specification of the program each machine should follow. In simpler situations where only two participants are involved, we will use other capital letters like  $A, B$  to denote participants.

Each machine has private input, output and random tapes, and one communication tape. The contents of the input tape of  $P_i$  is called  $ip_i$ , the contents of the output tape after execution of the protocol is called  $op_i$ . All machines are clocked by a common clock, and the protocol proceeds in *rounds*. In each round, exactly one machine can do some computation and perhaps write a message on the other machines' communication tapes. By convention, all messages sent are written simultaneously on all communication tapes, and the originator of any message can always be correctly identified. If a participant receives a message which does not obey the constraints specified in the protocol, it stops immediately, and we say that *cheating has been detected*.

All participants agree initially on a value of *security parameter*  $m$ .  $m$  is a measure of the amount of work that has to be done in the protocol, and also a measure of the cryptographic security.

The objective of the protocol can be described as computing one output for each participant, with a certain probability distribution. This distribution may depend on all the private inputs. For simplicity, we will specialize here to the case where the output is public, i.e. there is one output  $op$ , such that  $op = op_1 = \dots = op_n$ . We will discuss later the generalization to different (private) outputs. We let  $OP_I$  denote the probability distribution of  $op$ , when the inputs are  $I = (ip_1, \dots, ip_n)$ .

We assume of course that the desired computation is feasible to begin with, i.e. there is a not too large probabilistic boolean circuit which on inputs  $ip_1, \dots, ip_n$  produces an output distributed according to  $OP_I$ . It is unnecessary in this context to define precisely what "not too large" means. The precise meaning may depend on different practical circumstances, and it may be desirable to change the level of cryptographic security independently of the complexity of the task of the protocol. The only thing we require is therefore that the running time of the protocol must not grow too rapidly as a function of the size of the boolean circuit doing the computation.

The first demand to a protocol is of course that it computes correct results based on the inputs given. Before we can make this statement more precise, we have to address a technical problem, however: if some participants do not follow the protocol, we have no guarantee that there exists such a thing as a well defined input from these participants - they may succeed in



not behaving consistently with any choice of input. This problem can be solved by jumping ahead in the protocol descriptions to come later: we will require that any participant who has private input publishes some information which commits him to a particular choice of input (a set of bit commitments, for example). We may then simply *define* the input of each participant to be whatever was committed to during the protocol. Still there is no guarantee, however, that all participants will commit to the bit string they are given on the input tape - nothing can prevent a participant from pretending that the contents of its input tape is different from what in fact it is. This means, that to achieve the highest level of generality, we have to work with two levels of input, for  $P_i$  we need the original  $ip_i$ , and in addition,  $i\tilde{p}_i$ , denoting the bit string actually committed to during the protocol. In this thesis, however, we will restrict our attention to participants, for which  $ip_i$  always equals  $i\tilde{p}_i$ . In less technical terms: we will only look at participants who make up their mind about their choice of input before the protocol starts, and do not choose it, for example as some function of messages they see during the protocol. But as long as the protocol forces all participants to behave consistently with one choice of input, this restriction is only a technicality: it is intuitively reasonable that choosing input during the protocol does not help a participant to find other peoples' secrets or damage the result. Indeed, it turns out that the general situation can be handled by essentially the same protocol construction as given in Section 8, but this would cause considerably more complicated notation and definitions, whence we have chosen the simpler approach, to improve readability of the thesis.

With this in mind, we can return to the definition of correctness: Ideally, we would like the protocol to reproduce exactly the desired distribution,  $OP_I$  for all inputs  $I$ . This, however, turns out to be too restrictive in practice. It seems that in order to also ensure privacy, one must allow the participants a very small, but non zero chance of successful cheating. Thus the best we can do is to ensure that the output from the protocol is distributed *almost* as  $OP_I$ . Let  $OP_{I,m}^{prot}$  denote the probability distribution according to which the output of a participant following the protocol is distributed, when the protocol is executed with security parameter  $m$  and no cheating is detected. We let  $OP_I(op)$  (resp.  $OP_{I,m}^{prot}(op)$ ) denote the probability that the binary string  $op$  is produced as output. We would now like  $OP_I(op)$  to be almost equal to  $OP_{I,m}^{prot}(op)$ , at least asymptotically. If this is so, then the only way for a (set of) dishonest participant(s) to prevent correct results from being computed is to stop the protocol early e.g. by deliberately sending an incorrect message. More precisely, we have the following:

### Definition 6.1

A protocol is said to be *correct* if for any choice of inputs  $I = (ip_1, \dots, ip_n)$ , and any binary string  $op$  and constant  $c$ ,

$$|OP_I(op) - OP_{I,m}^{prot}(op)| < m^{-c},$$

for all sufficiently large  $m$   $\square$

We now come to the notion of security. We would of course like the protocol to keep the private inputs “as secret as possible”. It is clear, however, that no protocol can hope to do better than limiting the amount of communicated information to exactly what each participant is *entitled* to know, namely his own private input and the public output. Thus we will say that a protocol is secure, if whatever some coalition of participants can compute after execution of the protocol, they could also have computed *only* from their own private inputs and the public output. To state this formally, we need some definitions:

The binary string which is the concatenation (in correct time sequence) of all messages sent during the protocol is called a *protocol conversation*.

As mentioned, we will model the computation we want from the protocol by a probabilistic Boolean circuit. Let  $ra$  be a binary string containing as many bits as there are random inputs to the circuit. When the random inputs and all private inputs are specified, the output is of course uniquely determined, in other words, with private inputs  $ip_1, \dots, ip_n$  and random choice  $ra$ , the output is  $f(ip_1, \dots, ip_n, ra)$ , where  $f$  is an ordinary deterministic function.

We let  $P_{prot}^{(m)}(ip_1, \dots, ip_n, ra)$  denote the probability distribution according to which a protocol conversation with security parameter  $m$  is distributed. This distribution is taken over the coinflips of all participants. If some of the participants are *dishonest*, i.e. do not follow the protocol, the distribution also depends on the program followed by these participants.

A *conspiracy* is a subset  $X$  of  $\{P_1, \dots, P_n\}$  with  $|X| < n$ . The machines in a conspiracy may follow any polynomial time program, and they may establish private communication channels to share all information available to them.

A *simulator* for  $X$  is a polynomial time probabilistic Turing machine,  $M_X$ , which on input  $m, f(ip_1, \dots, ip_n, ra)$  and  $\{ip_i \mid P_i \in X\}$  will generate a protocol conversation, distributed according to the probability distribution  $P_{M_X}^{(m)}(f(ip_1, \dots, ip_n, ra), \{ip_i \mid P_i \in X\})$ .

## Definition 6.2

A protocol is said to be *secure against the conspiracy*  $X$ , if there exists a simulator for  $X$ ,  $M_X$ , such that for any choice of inputs  $ip_1, \dots, ip_n$  and random choice  $ra$ , the two ensembles of probability distributions

$$\{P_{M_X}^{(m)}(f(ip_1, \dots, ip_n, ra), \{ip_i \mid P_i \in X\})\}_{m=1}^{\infty}$$

and

$$\{P_{prot}^{(m)}(ip_1, \dots, ip_n, ra)\}_{m=1}^{\infty}$$

are polynomially (in  $m$ ) indistinguishable. A protocol is said to be *minimum knowledge*, if it is secure against any conspiracy. If the two ensembles of probability distributions are in fact equal for any conspiracy, the protocol is said to be *perfect minimum knowledge*  $\square$

Loosely speaking, two ensembles of probability distributions are polynomially indistinguishable if no probabilistic polynomial time algorithm can guess which of the two distributions a given binary string was drawn from. The formal details can be found in [GoMiRa].

Thus, a correct minimum knowledge protocol has the following properties: If the protocol completes with no cheating detected, then the result computed is (with very large probability) a correct one. Further, no conspiracy will be able to find out secrets of other participants, nor the random choices made in the computation, no matter what kind of hostile behavior it exhibits, including stopping the protocol in some way.

The definitions do not, however, exclude the possibility that a conspiracy could, by stopping the protocol in some clever way, find the output, while preventing honest participants from getting it. We will discuss later how to protect against this.

Very similar definitions have been proposed by Galil, Haber and Yung in [GaHaYu], the main difference being that in [GaHaYu], the simulator goes to an oracle and obtains a result distributed according to  $OP_I$ . It then has to simulate based on this and whatever private input it knows. Thus an explicit mentioning of the random choices in the computation is avoided. But since for many applications, it is essential that the random choices are kept secret by the protocol, it seems more natural that they are treated just like any other input to the computation, as far as security is concerned. We have therefore chosen the definitions given above.

Like [GaHaYu], we have chosen the word “minimum knowledge”, rather than “zero knowledge”, as was used in [GoMiRa]. This is because knowledge *is* in fact communicated in these protocols, namely the result of the computation. As explained above, the task is to limit the information communicated to a minimum. In [BrChCr], a third term is used, namely “minimum disclosure”. This refers to protocols where an adversary cannot after doing the protocol compute more about his opponents’ secrets, than he could before the protocol was executed. The difference here is that with minimum knowledge, we require that the adversary gets no advantage in computing anything *at all*.

For the understanding of the next section, it will be helpful to make explicit the connection between the formalism in this section and the original zero- knowledge definitions from [GoMiRa].

In [GoMiRa] the emphasis is on proof systems, i.e. protocols where a prover tries to convince a verifier that a given binary string  $x$  belongs to some language  $L$ . In the original model it was assumed that the prover had infinite computing power, while the verifier was restricted to polynomial time computations. Thus one difference in our work is that we assume that all participants have only bounded resources. This is the natural cryptographic situation, and it allows a more general use of bit commitment schemes. Apart from this difference, our definitions are just straightforward generalizations of the definitions of zero-knowledge proof systems to multiparty environments.

To see this, consider the case where  $L$  is an NP-language, and think of the prover as a polynomial time machine, which happens to know a certificate of membership in  $L$  for  $x$ ,

perhaps as a result of constructing  $x$  itself. In fact the prover and the verifier are now trying to do an interactive computation, where only the prover supplies input, namely the certificate for  $x$ . The computation simply consists of checking the certificate, which is easy by definition of NP. Therefore the notion of correctness corresponds to the fact that we have a proof system: we want the protocol to produce the correct output of the computation, namely “accept  $x$ ” if indeed the prover put in a valid certificate for  $x$ , and “reject” otherwise. In this situation, we will sometimes refer to the protocol as being *correct with respect to the statement  $x \in L$* .

As for minimum knowledge, note that conspiracies can only have size 1 in the two-party case, and that it does not make sense to consider the prover trying to violate the privacy of the verifier, because the verifier does not supply any private inputs. Thus the protocol is minimum knowledge according to definition 6.2 if the verifier can simulate the protocol for himself, assuming only that  $x$  does indeed belong to  $L$ . But this is exactly the original zero knowledge definition.

Thus saying that the protocol is correct and minimum knowledge, is in this case exactly equivalent to saying that it is a zero knowledge proof system.

## 7. Gradual and Verifiable Release of a Secret.

In this section, we present protocols allowing someone with a secret discrete logarithm to release it, bit by bit, such that anyone can verify each bit's correctness as they receive it. This new notion of *release* of secrets generalizes and extends that of the already known *exchange* of secrets protocols. Consequently, the protocols presented allow exchange of secret discrete logs between any number of parties.

The basic protocol solves an even more general problem than that of releasing a discrete log. Given any instance of a discrete log problem in a group with public group operation, the party who knows the solution can make public some interval  $I$  and convince anyone that the solution belongs to  $I$ , using a minimum knowledge protocol, so that no additional information is released (such as any hint as to where in  $I$  the solution is).

This can be used directly to release a discrete log, or to *transfer* it securely between different groups, i.e. prove that two instances are related such that knowledge of the solution to one implies knowledge of the solution to the other.

We show how this last application can be used to implement a more efficient release protocol by transferring the given discrete log instance to a group with special properties. In this scenario, each bit of the secret can be verified by a single modular squaring, and unlike the direct use of the basic protocol, no interactive proofs are needed after the basic setup has been done.

Finally, it is shown how the basic protocol can be used to release the factorization of a public composite number.

The results in this section were presented in part in [BCDG].

### 7.1. Related work.

Two of the first contributions to the field of *secrets exchange* were made by Blum[B1] and Rabin[Ra]. They both showed how to exchange secrets using oblivious transfer. Later, Blum[B12] proposed his well known protocol for exchange of factorizations; see Hastad and Shamir [HaSh] for a discussion of some problems with this protocol. It cleverly exploits the fact that the factorization of an integer is a secret of many bits. In the original version, one participant would sometimes know one secret bit more than the other. Tedrick's work [Te], [Te2] is one approach to reducing the amount of unfairness introduced by this fact. A different approach to exchanging one bit was taken by Luby, Micali and Rackoff [LuMiRa] and Vazirani and Vazirani [VaVa]. Here, the information exchanged is probabilistic, i.e. in each round of the protocol additional certainty is gained about the value of a single secret bit. The work of Yao [Ya] implies the existence of extremely general two party exchange-of-secret protocols, in connection with the problem of secure computations with private inputs.

While the practical proposals in the literature involve disclosure of secret factorizations (combined with square roots), the present protocol allows disclosure of discrete logarithms. This is of practical importance for protocols based on secret discrete logs, since means to



*transfer* a secret from a discrete log form to a factorization form are not known.

Perhaps a more fundamental difference between the related literature and the present protocol, however, is that they only provide for *exchange* of secrets, while we allow the more general *release* of secrets. One possible exception is [VaVa]. However, the fact that the information released by our protocol is deterministic, remains a fundamental difference between the protocols. In earlier exchange of secrets protocols, it was either inherent in the protocol that an exchange had to take place ([LuMiRa]), or the work done by the party with the secret would increase rapidly with the number of parties who were to receive it ([Bl2]). In a release protocol the party with the secret is essentially just broadcasting information about it. Therefore, a release can be simultaneous to any number of parties, and can be readily used to implement an exchange of secrets between any number of parties.

The theoretical existence of such release-of-secrets protocols can be proved using techniques from the work of Brassard and Crépeau [BrCr], of Chaum [Ch] and of Goldreich, Micali and Wigderson [GoMiWi2]. But since their methods (as Yao's [Ya]) involve individual processing of all gates in a Boolean circuit or a reduction to 3COL, the resulting protocols would be impractical. We propose a more practical solution to the problem: in the most efficient version, to release one bit, one modular square root has to be computed, while the correctness of that bit can be checked by one squaring.

## 7.2. Showing membership in an interval.

Suppose one party – the prover ( $P$ ) – knows a solution  $z$  to the equation  $\alpha^z = \beta$  where  $\alpha$  and  $\beta$  belong to some group with a public group operation. One obvious example of this is the multiplicative subgroup of the integers modulo  $p$ ,  $\mathbb{Z}_p^*$ , where  $p$  is a large  $m$ -bit prime. Suppose also that  $z \in [a, a+B]$ , where  $a$  and  $B$  are public. The protocol below can then be used by the prover to convince someone else – the verifier ( $V$ ) – that  $y \in [a-B, a+2B]$ .

Although we talk about a single verifier for convenience, it should be noted that any set of participants could easily play the part of  $V$  without losing the efficiency of the protocol. This is so because the role of  $V$  is passive: he flips coins (in public) and expects answers from  $P$  according to the coinflips. Thus, this protocol has the basic properties we require from a release of secrets protocol.

Note that since “there exists a solution to this discrete log problem which belongs to  $[a-B, a+2B]$ ” is certainly an NP-statement, we are here dealing with the special case of the general computation problem discussed at the end of the previous section.

In the protocol below,  $P$  will make use of a bit commitment scheme. Any of the examples outlined in Section 4 will do, although different choices will have different levels of security, as detailed below.

Having chosen an instance of size  $m$  of a bit commitment scheme, the prover can commit to a string of bits by simply computing commitments bit by bit. For the bit string  $s$ , the resulting string of commitments is denoted  $BC(s)$ . Any computation of bit commitments



involves of course a random choice, but in this section this has been removed from the notation for simplicity.

#### PROTOCOL *RELEASE INTERVAL*

1.  $P$  picks  $t_1 \in [0, B]$  and computes  $t_2 = t_1 - B$ .  
 $P$  computes a bit commitment of  $\alpha^{t_1}$  and of  $\alpha^{t_2}$ , and sends the unordered pair  $(BC(\alpha^{t_1}), BC(\alpha^{t_2}))$  to  $V$ .
2.  $V$  flips a coin and requests either
  - (a) to receive  $t_1$  and  $t_2$  and opening by  $P$  of both  $BC(\alpha^{t_1})$  and  $BC(\alpha^{t_2})$ , or
  - (b) to receive  $(z + t_i)$  for  $i = 1$  or  $2$ , and opening by  $P$  of  $BC(\alpha^{t_i})$
3. On request (a)  $P$  sends  $t_1$  and  $t_2$ , and opens both commitments.  
 On request (b)  $P$  sends either  $(z + t_1)$  or  $(z + t_2)$ , whichever is in the interval  $[a, a + B]$ , and opens the corresponding bit commitment.
4. On request (a),  $V$  checks that the two commitments did indeed contain  $\alpha^{t_1}$  and  $\alpha^{t_2}$ , that  $t_1 \in [0, B]$ , and that  $t_1 - t_2 = B$ .  
 On request (b),  $V$  checks that  $\alpha^{z+t_i}$  equals  $\beta\alpha^{t_i}$  (he knows  $\alpha^{t_i}$  as a result of the opening of one of the commitments).

Steps 1-4 are repeated  $m$  times.

#### Lemma 7.1

If in any of the  $m$  rounds, the prover cannot change the contents of his commitments after step 1, and is able to satisfy both request (a) and request (b), then  $z \in [a - B, a + 2B]$ .

**Proof.**

The result follows immediately from  $t_1 \in [0, B]$ ,  $t_2 \in [-B, 0]$ , and  $z + t_i \in [a, a + B]$  for either  $i = 1$  or  $2$ .  $\square$

#### Theorem 7.2

With respect to the statement  $z \in [a - B, a + 2B]$ , *RELEASE INTERVAL* is a correct protocol.

**Proof.**

Suppose that  $P$  has not been able to change the contents of any of his bit commitments during the protocol. Then the probability that  $P$  manages to satisfy  $V$  in all  $m$  rounds, if  $z$  is not in the specified interval is at most  $2^{-m}$ . But by unforgeability of the bit commitment scheme used, the condition is only violated in a superpolynomially small fraction of the cases  $\square$

Note that there is a discrepancy between the requirement for  $z$ , namely  $z \in [a, a + B]$ , and what is actually proven, namely  $z \in [a - B, a + 2B]$ . Below we will show that only if

$z \in [a, a+B]$ , the above protocol is minimum knowledge.

For this, we need the following lemma:

**Lemma 7.3**

If  $z \in [a, a+B]$ , then the distribution of the value of  $(z + t_i)$  sent in step 3(b) of RELEASE INTERVAL is independent of the value of  $z$ .

**Proof.**

It is trivial to check that if some number  $d \in [a, a+B]$  is sent as  $(z + t_i)$ , then exactly one value in  $[0, B]$  must have been chosen as  $t_1$ . So if  $t_1$  is chosen uniformly in  $[0, B]$ , then  $(z + t_i)$  must be uniform in  $[a, a+B]$   $\square$ .

**Theorem 7.4**

If  $z \in [a, a+B]$ , then RELEASE INTERVAL is a minimum knowledge protocol.

**Proof.**

By the remarks at the end of Section 6, it suffices to exhibit a simulator  $M_V$ , which will simulate the protocol given  $V$ ,  $\beta$  and the fact that  $z \in [a, a+B]$ . For this, we use the following standard trick from the theory of minimum knowledge protocols: since the protocol proceeds in rounds, the simulator can rewind  $V$  (i.e. stop and go back to a previous state) if the simulation gets stuck at any point. Thus, we can essentially assume that the simulator knows in advance which choice the verifier will make in step 2. It is therefore trivial to compute a message for step 1 that will satisfy the verifier if he is going to make choice (a). For choice (b) the simulator chooses  $d \in [a, a+B]$  at random, to serve as  $z + t_i$ . It then puts  $\gamma = \alpha^d \beta^{-1}$ . Now  $M_V$  can commit to  $\gamma$ , and to something totally random that has the same number of bits as  $\gamma$ . These commitments are sent in step 1; in step 3,  $d$  is sent in place of  $z + t_i$ , and the bit commitment containing  $\gamma$  is opened.

If bit commitments based on claw free functions are used, it is now clear from Lemma 7.3 that the simulated conversation will have exactly the same distribution as the actual protocol conversation, so that the protocol is perfect minimum knowledge in this case.

If commitments based on probabilistic encryption are used, the distributions will be different. The only difference, however, lies in the distribution of bits hidden in commitments, so by the Hiding property of bit commitment schemes, the difference cannot be recognized in polynomial time  $\square$

$P$  is of course free to choose  $z \in [a+B, a+2B]$ , but then the protocol releases information. For instance, if  $z$  is almost equal to  $a+2B$ ,  $P$  must always choose  $t_1$  close to 0, and release  $(z+t_2)$  which is close to  $a+B$ . We could, however, take away  $P$ 's freedom to choose  $t_1$  himself, and in this way get a proof that  $z$  is contained in a smaller interval. The protocol could start by a sequence of coinflips with the property that both parties trust their

randomness, but only  $P$  gets to see the outcome. This can easily be implemented using bit commitments:

### PROTOCOL RELEASE INTERVAL II

- i.  $P$  chooses  $r$  independent random bits  $b_1, \dots, b_r$ , and computes a set  $BC(b_1, \dots, b_r)$  of commitments to these bits, which he sends to  $V$ .
- ii.  $V$  chooses and sends to  $P$   $r$  independent random bits  $v_1, \dots, v_r$ .
1.  $P$  computes  $t_1 = g(b_1 \oplus v_1, \dots, b_r \oplus v_r)$ , and  $t_2 = t_1 - B$ .  $g$  is a fixed function agreed on initially with the property that when its arguments are  $r$  independent random bits, the result is a randomly chosen element in  $[0, B]$ . He then computes a bit commitment of  $\alpha^{t_1}$  and of  $\alpha^{t_2}$  and sends the unordered pair  $(BC(\alpha^{t_1}), BC(\alpha^{t_2}))$  to the verifier.
2.  $V$  flips a coin and requests either
  - (a) to receive  $t_1$  and  $t_2$  and opening by  $P$  of  $BC(\alpha^{t_1})$ ,  $BC(\alpha^{t_2})$  and  $BC(b_1, \dots, b_r)$ , or
  - (b) to receive  $z + t_i$  for  $i = 1$  or  $2$  and opening by  $P$  of  $BC(\alpha^{t_i})$ .
3. On request (a)  $P$  sends  $t_1$  and  $t_2$ , and opens all his commitments.  
On request (b)  $P$  sends either  $z + t_1$  or  $z + t_2$ , whichever is in the interval  $[a, a + B]$ , and opens the corresponding bit commitment.
4. On request (a),  $V$  checks that the two commitments did indeed contain  $\alpha^{t_1}$  and  $\alpha^{t_2}$ , that  $t_1 - t_2 = B$ , and that  $t_1 = g(b_1 \oplus v_1, \dots, b_r \oplus v_r)$ .  
On request (b),  $V$  checks that  $\alpha^{z+t_i}$  equals  $\beta \alpha^{t_i}$  (he knows  $\alpha^{t_i}$  as a result of the opening of one of the commitments).

Steps i. to 4. are repeated  $n$  times.

Unless  $B$  is a power of 2, the function  $g$  cannot be designed to output a uniformly chosen element in  $[0, B]$ . We can, however, make a very close approximation, and therefore the proof of Theorem 7.5 below works under the assumption that the output of  $g$  is uniformly distributed. The reader can easily work out for himself the (trivial) changes needed in the proof to make up for the deviation from a uniform distribution.

### Theorem 7.5

If RELEASE INTERVAL II is used with a number of rounds  $n \geq m \frac{1}{-\log_2(1-\epsilon/2)}$ , then it is a correct protocol with respect to the statement  $z \in [a - \epsilon B, a + (1+\epsilon)B]$ .

**Proof.**

Consider the case where  $P$  is trying to cheat, i.e.  $z \notin [a - \epsilon B, a + (1+\epsilon)B]$ , and let us find the probability that  $P$  manages to satisfy  $V$  in one round of the protocol. There are two cases:

- 1) The  $t_1$  selected in steps i. and ii. is such that  $P$  cannot answer both request (a) and (b) in

step 3. In this case,  $P$ 's chance of success is at most  $1/2$ .

2) The  $t_1$  selected in steps i. and ii. is such that  $P$  can answer both request (a) and (b). In this case,  $P$ 's chance of success is of course 1.

Since case 1) is easily seen to occur with probability  $\varepsilon$ , if  $V$  plays correctly,  $P$ 's overall chance of success in one round is  $\varepsilon \cdot 1/2 + (1-\varepsilon) \cdot 1 = 1 - \varepsilon/2$ , and therefore  $P$ 's chance of surviving  $n$  rounds is  $(1-\varepsilon/2)^n$ . The protocol will be correct if this probability is exponentially small in the security parameter  $m$ . We therefore get the result on  $n$  by solving

$$(1-\varepsilon/2)^n = 2^{-m}$$

for  $n$   $\square$

This result tells us how many more rounds we need to get the same level of confidence for  $V$  with a smaller interval, as a function of the size of that interval. Note that for small  $\varepsilon$  we have that  $n$  is proportional to  $\frac{2m}{\varepsilon}$ , which shows that even if we allow  $\varepsilon$  to drop polynomially as a function of  $m$ , we still get a polynomial running time for the protocol.

#### Theorem 7.6

RELEASE INTERVAL II is a minimum knowledge protocol, if  $z \in [a, a+B]$ .

**Proof.**

Is almost identical to the proof of Theorem 7.4, and is therefore left to the reader. Once again, the protocol is perfect minimum knowledge is bit commitments based on claw free functions are used  $\square$

There is a third variation on the protocol, which does not require the use of bit commitments. It is slightly more efficient for that reason, but unfortunately the proof of minimum knowledge becomes very long and technical. Furthermore, it cannot be modified to be perfect minimum knowledge. For details, refer to [BCDG].

### 7.3. Two applications.

It is easy to see, how the RELEASE INTERVAL protocol can be used to release information about the solution to a discrete log problem in a verifiable way, starting with the high order bits: Use protocol RELEASE INTERVAL many times with ever decreasing values of  $B$ . This convinces the verifier that the solution is contained in smaller and smaller intervals.

If it is easy to compute square roots in the group involved, we can also release the solution, starting with the low order bits. Consider for instance  $Z_p^*$ , with  $p$  prime. Fix some generator  $\alpha$  of  $Z_p^*$ . For any square  $\beta \in Z_p^*$ , we let the principal square root of  $\beta$  be the root with discrete log base  $\alpha$  smaller than  $\frac{p-1}{2}$ . It is easy to see that the discrete log of any number can be computed bit by bit, starting with the low order bits, given an oracle that decides which of

two square roots is the principal one. At each point in this computation, someone who already knows the discrete log could serve as the oracle by using RELEASE INTERVAL to prove the validity of his answers. This clearly leads to a protocol that releases the discrete log starting with the low order bits. The “fuzzyness” at the borders of the intervals used means that we must use intervals slightly smaller than what is required by the above. This will result in a protocol that sometimes releases a little more than one bit at a time.

#### 7.4. An Efficiency Improvement.

In the previous two applications we had to go through the RELEASE INTERVAL protocol once for each bit released. This can be quite time consuming in general because each instance of RELEASE INTERVAL involves an interactive proof with many iterations. Using a technique to transfer a discrete log from one group to another, we can reduce this to 1 application of RELEASE INTERVAL. However, besides the protocol needed to transfer the logarithm, we need other protocols to prove that the groups used have specific properties. Recall that the secret  $z$  is determined by  $\alpha^z = \beta$ , where  $\alpha, \beta \in \mathbb{Z}_p^*$ . Basically we transfer the secret from  $\langle \alpha \rangle$  to  $\langle \alpha_1 \rangle$ , where  $\alpha_1 \in \mathbb{Z}_N^*$  and  $N$  is a composite.  $\alpha_1$  is chosen, such that  $2^l$  divides  $\text{order}(\alpha_1)$ , where  $l = \log_2(p)$ . Then the secret is released just by showing square roots in  $\mathbb{Z}_N^*$ , which are easily verified but hard to compute without the factorization. First we will show how  $P$  can prove to  $V$  that  $N$  has the required properties. This is done in steps 1-3 below. Note that these steps are only necessary once, i.e. if many secrets are to be released, the same  $N$  and  $\alpha_1$  can be used for all of them.

##### Step 1:

Party  $P$  chooses a modulus  $N = qr$ , where  $q$  and  $r$  are large primes, constructed such that  $P$  knows the factorization of  $q-1$  and  $r-1$ , and such that  $2^l$  divides at least one of  $q-1$  and  $r-1$ . This can easily be done, e.g. by a variation of Gordon’s method [Gor].

##### Step 2:

Let  $QR$  denote the set of quadratic residues modulo  $N$ . Next  $P$  selects a random element  $\alpha_1 \in \mathbb{Z}_N^*$  with  $(\frac{\alpha_1}{N}) = 1$  and  $\alpha$  not in  $QR$ . From the Chinese Remainder Theorem it can easily be seen that such an  $\alpha_1$  will always have the maximum possible 2-power dividing its order, so  $2^l$  is certainly a divisor of the order of  $\alpha_1$ .

##### Step 3:

$P$  makes public  $N$  and  $\alpha_1$ . Then

- a)  $P$  proves that  $\alpha_1$  is a quadratic non-residue, e.g. by using the protocol in [GoMiRa].
- b)  $P$  proves that  $2^l$  divides  $\text{order}(\alpha_1)$ :
- i)  $V$  chooses a random odd number  $r$ , with  $0 < r < R$ , where  $R$  is fixed (the choice of  $R$  is considered later).  $V$  chooses  $b \in \{0,1\}$ .

- ii)  $V$  sends to  $P$ :  
 $\alpha_1^{r \cdot 2^k}$  if  $b = 1$ ;  
 $\alpha_1^{r \cdot 2^{k-1}}$  if  $b = 0$ .
  - iii) Since  $P$  has constructed  $N$ , he can compute the order of what he receives. Based on the outcome he determines  $b$  and sends a commitment to  $b$ ,  $BC(b)$  to  $V$ .
  - iv)  $V$  sends  $r$  to  $P$ , who checks that it is correct, and then opens  $BC(b)$ .
  - iv)  $V$  checks that the correct value of  $b$  is contained in  $BC(b)$ .
- Steps i through iv are repeated  $n$  times.

It is easy to see that if  $P$  is not cheating, he can always give the correct answer in step iii). Consider the situation where  $P$  is trying to cheat, i.e. the maximum 2-power dividing  $order(\alpha_1)$  is  $2^c$ , where  $c < l$ . Let  $G$  be the maximal sub-group of  $\langle \alpha_1 \rangle$  of odd order. Clearly,  $G = \langle \alpha_1^{2^k} \rangle = \langle \alpha_1^{2^{k-1}} \rangle$ , which means that both  $\alpha_1^{r \cdot 2^k}$  and  $\alpha_1^{r \cdot 2^{k-1}}$  will look to  $P$  like randomly chosen elements of  $G$ . Since  $V$  does not know the order of  $G$ , he cannot make the distribution completely uniform over  $G$ , but he can make as close an approximation as he likes by choosing the  $R$  from step i) large enough. Assuming that  $P$  cannot distinguish the distribution of  $\alpha_1^{r \cdot 2^k}$  and  $\alpha_1^{r \cdot 2^{k-1}}$ , he can do no better than guessing at random in step iii) whence he will be caught with probability  $1 - 2^{-n}$ .

Furthermore it is intuitively clear that  $V$  learns nothing from this protocol, other than the fact that  $2^l$  divides the order of  $\alpha_1$ , simply because  $P$  never answers anything, unless  $V$  already knows the answer. It is easy to formalize this with a simulation argument. Thus we have:

#### Lemma 7.7

The protocol in Step 3 is minimum knowledge and correct with respect to the statement “ $2^l$  divides  $order(\alpha_1)$ ”.

#### Step 4:

$P$  transfers the problem from  $Z_p^*$  to  $Z_N^*$ . Remember  $P$ 's secret is the solution  $z$  of  $\alpha^z = \beta_1$  in  $Z_p^*$ . Now  $P$  computes  $\beta_1 = \alpha_1^z$  in  $Z_N^*$  and makes  $\beta_1$  public. Using RELEASE INTERVAL in the cross-product of  $\langle \alpha \rangle$  and  $\langle \alpha_1 \rangle$ ,  $P$  can prove that he knows a  $z \in [1, p-1]$ , which solves both  $\alpha^z = \beta$  and  $\alpha_1^z = \beta_1$ .

#### Step 5:

To release a single bit (the least significant bit of  $z$ ),  $P$  proceeds as follows:

- if  $z = 2z'$ , he makes public a square root of  $\beta_1$ .
  - if  $z = 2z' + 1$ , he makes public a square root of  $\beta_1 \alpha_1^{-1}$ .
- Replace  $z$  by  $z'$ , and  $\beta_1$  by the square root released.



This step works because of the following two easily verified facts:

1. The  $l$  least significant bits of  $z$  are uniquely determined by the equation  $\alpha_1^z = \beta_1$ .
2. If  $\alpha_1$  is a non square mod  $N$  and  $\alpha_1^z = \beta_1$ , then  $z$  is even if and only if  $\beta_1$  is a square mod  $N$ .

By fact 1, the pair  $(\alpha_1, \beta_1)$  determines exactly one secret of size  $l$  bits, and by fact 2, anyone can check on  $P$  by squaring the numbers released.

Regarding the security of this protocol, notice first that steps 2 and 3 can be executed in minimum knowledge. Moreover, the release of  $\alpha_1$  cannot endanger the factorization of  $N$ , since anyone can select at random a number of Jacobi symbol 1 mod  $N$ , which with probability 1/2 will have the same properties as  $\alpha_1$ . It is clear that breaking this scheme can be no harder than factoring numbers of the required special form:  $N = qr$ , where  $2^l$  divides at least one of  $q-1$  and  $r-1$ . It is not known, however, whether these numbers are easier to factor than randomly chosen RSA-moduli. But with respect to the factoring algorithms known at present, the factorization of  $N$  should be safe, if  $q-1$  and  $r-1$  contain large prime factors, in addition to the 2-powers. Finally, it is also conceivable that someone could try to attack the equations  $\alpha^z = \beta$  and  $\alpha_1^z = \beta_1$  directly, without factoring  $N$ . But this involves solving a discrete log problem, and we know of no argument indicating that discrete log should be easier in this special case than in the general situation.

### 7.5. Releasing a factorization.

This final application allows a prover to release the factorization of a public modulus  $N$ . It is wellknown that knowledge of a multiple of  $\phi(N)$  suffices to factor  $N$  easily. Using this fact, the protocol proceeds as follows: The verifier chooses an integer  $T$  with  $N \leq T \leq 2N$  and a set of elements  $a_1, \dots, a_s$  in  $\mathbb{Z}_N^*$ . These numbers are sent to the prover. The prover then computes  $t = T \bmod \phi(N)$ , and  $b_i = a_i^t$ . The verifier can also compute the  $b_i$ 's by raising the  $a_i$ 's to the  $T$ 'th power. The prover then uses the RELEASE INTERVAL protocol to convince the verifier that he knows a simultaneous solution to all  $s$  discrete log instances which belongs to an interval NOT including  $T$ . When the prover uses RELEASE INTERVAL more times to release bits of  $t$ , the verifier can be convinced, that he will get at least a multiple of  $LCM(\text{order}(a_1), \dots, \text{order}(a_s))$ . The probability that this number equals the exponent of the group  $\mathbb{Z}_N^*$  (i.e. the smallest integer  $e$  such that  $g^e = 1$  for all elements in the group) is exponentially large in  $s$ , and it is not hard to see that  $\text{exponent}(\mathbb{Z}_N^*)$  will serve as well in factoring  $N$  as  $\phi(N)$ .

Thus the protocols presented here are able to replace all the old "exchange of factorisation" protocols, with the additional improvement that the new protocol is provably minimum knowledge. Also the 1-bit exchange problem of [LuMiRa] can be dealt with, with a little extra work: The problem is how to slowly release the quadratic character of a number of Jacobi symbol 1 modulo a Blum integer  $N$ , or in our terminology: how to slowly open a bit commitment  $BC(b, r)$  constructed with QRS.

A trivial solution is to just release the factors of  $N$ . But this would reveal all other bits that may have been hidden with this commitment scheme. So a better solution is to create a new Blum integer  $N'$  and a commitment  $BC(b, r')$  modulo  $N'$ . Using the COMPARE COMMITMENTS protocol from Section 5, one then proves that the new commitment also contains  $b$ , and finally use the protocol above to release the factorisation of  $N'$ , with which one can easily compute  $b$  from  $BC(b, r')$ .

## 8. Multiparty Computations.

In this section, we present a protocol that allows a set of parties to collectively perform any agreed computation, where each party is able to choose secret inputs and verify that the resulting output is correct, and where all secret inputs are optimally protected.

The protocol has the following properties:

- One participant is allowed to hide his secrets *unconditionally*, i.e. the protocol releases no Shannon information about these secrets. This means that a participant with bounded resources can perform computations securely with a participant who may have unlimited computing power. To the best of our knowledge, our protocol is the first general computation protocol to provide this possibility.
- The cost of our protocol is linear in the number of gates in a circuit performing the computation, and in the number of participants. We believe it is conceptually simpler and more efficient than other protocols solving related problems ([Ya], [GoMiWi] and [GaHaYu]). It therefore leads to practical solutions of problems involving small circuits: with special-purpose hardware for modular arithmetic and using 512-bit numbers, one can process a few hundred binary gates per second while allowing a probability of less than  $2^{-100}$  of successful cheating.
- The protocol is *openly verifiable*, i.e. any number of people can later come in and rechallenge any participant to verify that no cheating has occurred.
- The protocol is minimum knowledge: even if  $n-1$  out of the  $n$  participants collude, they will not find out more about the remaining participants' secrets than what they could already infer from their own input and the public output.
- Each participant has a chance of undetected cheating that is only exponentially small in the amount of time and space needed for the protocol.
- The protocol adapts easily, and with negligible extra cost, to various additional requirements, e.g. making part of the output private to some participant, ensuring that the participants learn the output simultaneously, etc.
- Participants can prove relations between data used in different instances of the protocol, even if those instances involve different groups of participants. For example, it can be proved that the output of one computation was used as input to another, without revealing more about this data.
- The protocol can be used as an essential tool in proving that all languages in IP have minimum knowledge proof systems, i.e. any statement which can be proved interactively can also be proved in minimum knowledge. The proof avoids using the Goldwasser-Sipser result on equivalence between interactive proofs and Arthur-Merlin games. This is interesting in particular because it shows that the modification of an IP-protocol to a minimum knowledge protocol need only cost a polynomial factor in running time.

The results in this section are based on the material presented in [ChDaGr].

## 8.1. Related Work.

The problem of multiparty secure computations is an important one. When stated in its most general form, its solution implies a solution - at least a theoretical one - to almost any protocol construction problem. The first to consider this problem was Yao [Ya2]. He devised protocols, that under various cryptographic assumptions could keep the inputs secret while allowing each participant only a negligible chance of cheating. He did not, however, address the problem of “fairness”, i.e. ensuring that the participants learn the output simultaneously. Later, Yao [Ya] solved also this problem for the two-party case. Recently, in work done independently from (but earlier than) ours, Goldreich, Micali and Wigderson [GoMiWi] used Yao’s two-party construction to devise a multiparty solution, based on the existence of a trapdoor one-way function. This protocol implements a multiparty simulation of the computation in a circuit. Each participant holds a share of each bit in the real computation, and these shares are manipulated by using Yao’s two party construction  $O(n^2)$  times, where  $n$  is the number of participants. In other work done independently from and almost simultaneously with ours, Galil, Haber and Yung [GaHaYu] generalise all the properties of Yao’s construction to the multiparty case, and simplify the use of Yao’s protocol in the multiparty simulation. Also, they give a concrete construction based on Diffie-Hellman key exchange rather than the existence of a trapdoor one way function. Very recently, Goldreich and Vainish [GoVa] found another simplification by designing a special purpose two-party protocol which could replace Yao’s protocol in the multiparty construction. While this protocol is quite simple and efficient, it does not provide protection against cheating - further complications have to be introduced to achieve this.

In comparison, our protocol solves the multiparty case using a totally different concept. It attacks the problem “directly”, without using any two-party subprotocols, and without using reductions to 3-colorability [GoMiWi2] or to SAT [BrCr][Ch] to prove the validity of messages. We rely on a stronger cryptographic assumption than [GoMiWi], namely the *Quadratic Residuosity Assumption* [GoMi]. This, however, allows a simpler and more efficient construction, capable of implementing all the “primitives for cryptographic computations”, defined in [GaHaYu]. Although our protocol could also be based only on the existence of an arbitrary trapdoor one-way permutation, this would reduce the efficiency considerably. Some of the techniques, upon which our protocol is based, were developed and used in the work of Chaum [Ch] and Brassard and Crépeau [BrCr]. By specializing to the case where only one participant supplies the input, our protocol can provide slightly more efficient solutions to the problems solved there.

An interesting consequence of our work is that any statement which can be proved interactively, can also be proved in minimum knowledge, provided that our cryptographic assumption holds (one assumption which will do in this case is the existence of clawfree trapdoor functions). More formally: any language in IP has a minimum knowledge proof system.

This result was already proved for protocols with a constant number of rounds in [GoMiWi2], where a proof of the result in full generality is attributed to Ben-Or. Ben-Or's proof uses the result by Goldwasser and Sipser about the equivalence between interactive proofs and Arthur-Merlin games [GoSi]. Our proof eliminates the need for this result, at the cost of a stronger intractability assumption. The protocol of [GoSi] may require a prover to use infinite computing power, while we show that any interactive proof can be simulated in minimum knowledge with an increase in running time of only a polynomial factor. This means that our result is also interesting with respect to practical protocols.

## 8.2. Overview of the Construction.

In this section, we will give an informal description of the protocol. It turns out to be convenient to make some simplifications for this: we will consider only two parties,  $A$  and  $B$  and we will do a computation on one AND-gate only. At the end of the section we will see how this method can be generalised to more parties, and to a computation consisting of many gates.

Note that this AND-gate computation, where both parties want to hide their input from each other, has a meaningful application: consider the situation where Alice and Bob have just met, and each considers dating the other. Neither wishes to lose face in the following sense: if Alice wants a date but Bob doesn't, Alice does not want to let Bob know that she wanted the date. And the same holds for Bob. In other words: if a party does not want the date it does not find out the other party's decision.

Note that sometimes a party can derive the other's input just by combining his own input and the output of the joint computation. This can never be avoided, however: as usual we cannot deprive the parties of what they can compute from data they are entitled to know. Therefore what we are saying is essentially that we want a correct and minimum knowledge protocol for this problem.

As the first step in the protocol, both parties will choose a bit commitment scheme (Section 5). It is not necessary here to specify which commitment schemes are used, except that the one chosen by  $A$  must satisfy the blinding property, and both of them must satisfy the comparability property.

The AND-gate is represented by a truth-table  $T$ . We will show what transformations  $A$  and  $B$  have to apply to  $T$  in order to compute the logical AND, such that

- both parties can be sure (with very high probability) that the transformations are done correctly;
- both parties can keep their input hidden from the other party; and
- both parties can be sure the result of the computation is correct.



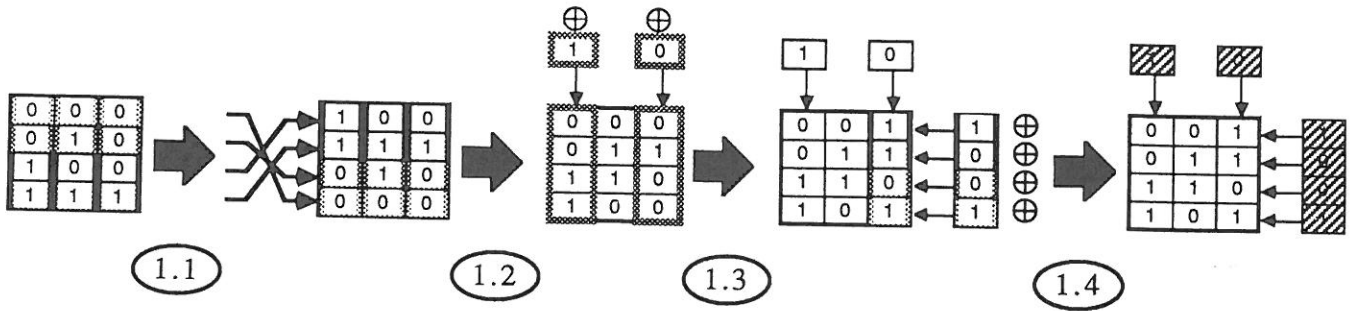


Figure 1. Party A's transformation of the AND-gate.  
 Note that all figures show the table *after*  
 the indicated operation has been performed.

#### THE PROTOCOL

- 1.1 A applies a randomly chosen permutation,  $\sigma_j$ , to the rows of  $T$ .
- 1.2 A chooses two bits  $b_1$  and  $b_3$ , and uses these to compute the XOR of the  $k$ th column and  $b_k$ . This procedure is known as *complementation of columns*.
- 1.3 A chooses four bits  $d_1 \cdots d_4$ , and computes the XOR of the  $l$ th entry in the last column and  $d_l$ . This procedure is known as *encrypting the output column*.
- 1.4 A computes commitments to  $b_1, b_3, d_1 \cdots d_4$  (see Section 5) and assigns the commitments to the rows or columns they correspond to. The commitments are shown on the figures as shaded boxes.

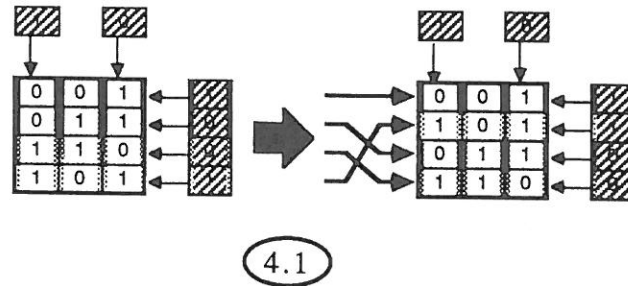
We will denote the final result of these steps by  $\hat{T}$ .

$B$  will verify whether  $\hat{T}$  was constructed correctly in the next steps:

2. A creates a transformed truthtable  $T'$  in exactly the same way as he created  $\hat{T}$ .
3.  $B$  flips a coin.
  - If the coin came out heads, A must reveal exactly how she constructed  $T'$ , so that  $B$  can verify that this was done correctly.
  - If the coin came out tails, A must show a special relation between  $\hat{T}$  and  $T'$ . Details on this can be found in Section 8.4.1. Here, it suffices to know that if the relation holds, it implies that  $\hat{T}$  was correctly constructed if and only if  $T'$  was.

After  $m$  repetitions of steps 2 and 3,  $B$  is convinced with very high probability that A constructed  $\hat{T}$  correctly. Now  $B$  will do to  $\hat{T}$  what A did to  $T$ , and even more:



Figure 2. The row-permutation performed by  $B$ .

- 4.1  $B$  permutes the rows of  $\hat{T}$ , together with the commitments of the output column entries.
- 4.2  $B$  flips a coin for each output column entry and applies the XOR.
- 4.3  $B$  computes, using  $A$ 's bit commitment scheme, a commitment containing the XOR of a bit chosen by him and the one chosen and committed to by  $A$  for this output column entry. This is where the blinding property of  $A$ 's commitment scheme is used. The

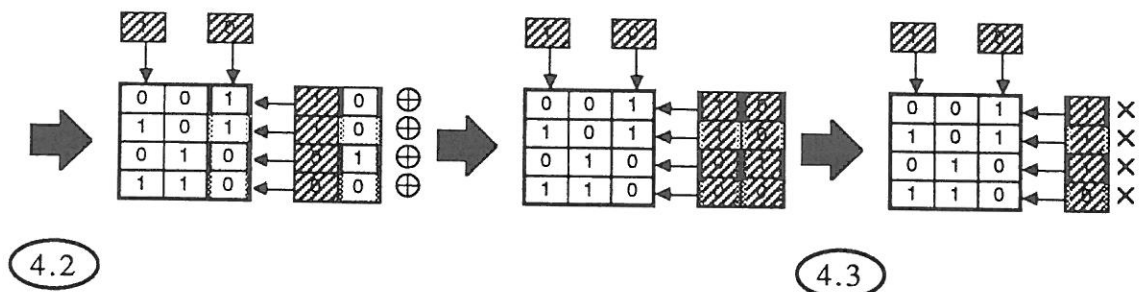


Figure 3. The blinding.

effect is that although  $A$  can later open the resulting commitment, he will have no idea which of his original commitments it was computed from.

This *blinding* is necessary to hide  $B$ 's permutation.  $B$  also has to do a transformation corresponding exactly to the one of  $A$ :

4.4  $B$  chooses bits for inversion of his own input column, and for the output column.

4.5  $B$  chooses bits for inversion of each output column entry.

4.6  $B$  commits to the bits chosen in steps 4.4 and 4.5. He then sends his final result,  $T^*$ , to  $A$ .

5. In the same way as in steps 2 and 3,  $B$  convinces  $A$  that  $T^*$  was constructed correctly.

Now  $A$  and  $B$  have together constructed a double-blinded version  $T^*$  of the gate that satisfies the conditions listed before step 1. They will use  $T^*$  in performing the computation as follows:

6. Together they select the correct row of  $T^*$  by announcing their choice of input bits for  $T^*$ . These input bits are the xor of the "real" input bits and the inversion bits for the corresponding columns.

7.  $A$  and  $B$  both reveal the inversion bit for the output column, and the inversion bit for the entry in the output column which is in the selected row. By x-oring this output column entry with the revealed inversion bits, they can both compute the bit that is the result of the computation.

This basic protocol can be generalized in several ways:

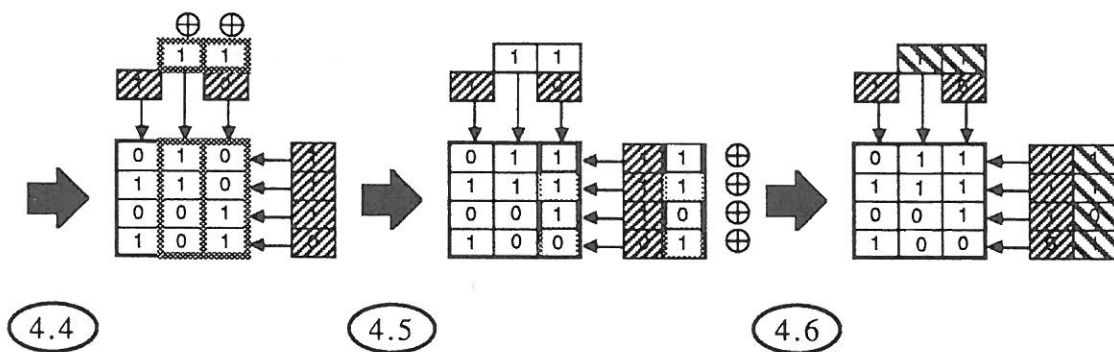


Figure 4. The final part of  $B$ 's transformation.

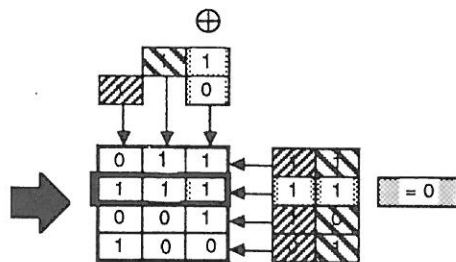


Figure 5. The computation. Here, both  $A$ 's and  $B$ 's input is 0, hence the output is also 0.

- Instead of using an AND-gate, any type of gate can be used.
- Instead of two parties, any number of parties can participate in the protocol. Each new party just has to follow  $B$ 's part of the protocol, and will thus have to blind the commitments of each participant who went before him.

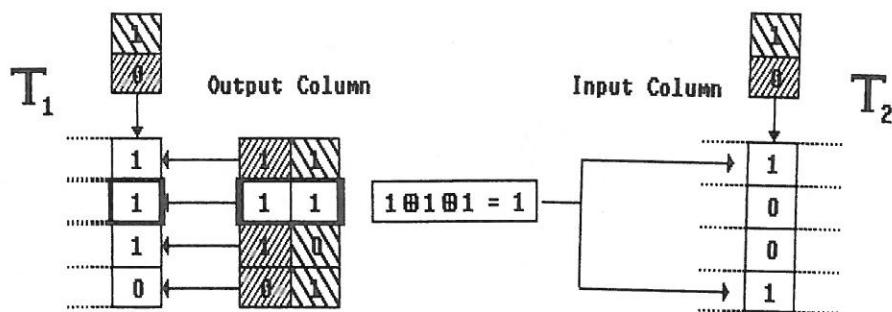


Figure 6. Connection between gates.

- Instead of using only one gate, a computation involving arbitrarily many gates can be done with the protocol. The only condition is that when gate  $T_1$  is connected to  $T_2$ , then their corresponding output column and input column were inverted using the *same* bits. When later a row is selected in the encrypted version of  $T_1$ , the inversion bits that apply to the single output column entry will be revealed, while the column inversion bits will remain secret. The revealed inversion bits are now x-ored with the output column entry selected, and the result can be used directly as an input bit to the encrypted version of  $T_2$  (see Fig. 6).

### 8.3. Circuits, terminology and notation.

We think of a boolean circuit  $C$  in the usual way as an acyclic directed graph, the nodes are called *gates*, the edges are called *wires*.

Some wires are connected only *into* one or more gates. These wires are called *input wires*. If it leaves a gate without connecting into another, we call it an *output wire*.

The input wires are partitioned into  $n+1$  disjoint subsets,  $I_1, \dots, I_n, R$ .  $I_j$  corresponds to the secret input chosen by  $P_j$ , such that  $ip_j$  contains one bit for each wire in  $I_j$ . In a similar way,  $R$  corresponds to the random inputs to  $C$ .

For each gate, a function is specified that maps the input bits to a one bit output. For the gate  $G$ , this function is given by *the truth table*  $T(G)$ .  $T(G)$  is a 0–1 matrix with  $t+1$  columns and  $2^t$  rows, where  $t$  is the number of wires connecting into  $G$ . The rows, apart from the last entry, contain each possible assignment of bits to the input of  $G$  exactly once, and the last column contains the corresponding output bits. Note that each input column in each truth table corresponds to exactly one wire in  $C$ , and that an output column may correspond to any number of wires (“fan-out” from a gate is allowed). Thus, two columns may correspond to the same wire. Two such columns are said to be *connected*.

For the sake of generality, we have designed the protocol to allow gates with arbitrary fan-in. In practice, however, it will always be advisable to use fan-in 2, since the work done in the protocol increases exponentially with the fan-in.

#### Definition 8.1.

A *computation*  $\Gamma$  in  $C$  consists of a *selection* of exactly one row in each truth table. The *input* of  $\Gamma$  is the corresponding assignment of bits to the input wires of  $C$ , the *output* of  $\Gamma$  is the corresponding assignment of bits to the output wires.

A computation is said to be *consistent*, if the following is satisfied: whenever two truth-table-columns  $c_1$  and  $c_2$  are connected, the entry selected by  $\Gamma$  in  $c_1$  equals the entry selected in  $c_2$   $\square$

### 8.3.1. Transformations of Truthtables.

In our protocol, all participants get to encrypt all truthtables in the circuit. This is done according to the numbering of the participants  $P_1, \dots, P_n$ : first  $P_1$  encrypts the circuit, sends his result to  $P_2$ , who does his encryption, etc. until  $P_n$  has done his part of what we call “the encryption phase”. The following definition of an  $i$ -transform describes what a truthtable should look like after it has been processed by  $P_i$ .

#### Definition 8.2.

Let  $G$  be a gate in  $C$  with  $t$  input wires. For  $i=1 \dots n$ , we define an  $i$ -transform  $T$  of  $T(G)$  to be a  $(t+1)$  by  $2^t$ , 0–1 matrix with a list of bit commitments attached to each row and each column. The list attached to row  $l$  is called  $rlist(l, T)$ . The list attached to column  $k$  is called  $clist(k, T)$ .

Each list contains one commitment from each participant  $P_j$  for which  $j \leq i$ . The only exception is  $clist(k, T)$ , when the  $k$ 'th column corresponds to an input wire in some  $I_j$ . These lists are empty if  $j > i$ , and contain one bit commitment from  $P_j$  if  $j \leq i$ .

By  $rsum(l, T)$  (resp.  $csum(k, T)$ ) we denote the x-or sum of all bits committed to in  $rlist(l, T)$  (resp.  $clist(k, T)$ ). The sum of an empty list is defined to be 0.

A 0-transform of  $T(G)$  is defined to be simply  $T(G)$  itself  $\square$

We require that there is some specific relation between a truthtable  $T(G)$  and its transforms used in the protocol. When this relation holds, the transform is said to be *valid*. In general, an  $i$ -transform is the product of work done by  $P_1, \dots, P_i$ , and the transform will be valid if all these participants have followed the protocol. A valid transform can be seen as an encrypted version of  $T(G)$ : the rows are permuted, and each column is x-ored with independently chosen bits. In addition to this, the last entry in each row is x-ored with other independently chosen bits. The attached lists of bit commitments (when opened) contain complete information about how the transform was constructed. More specifically we have the following:

#### Definition 8.3.

A *valid*  $i$ -transform  $T$  of  $T(G)$  is an  $i$ -transform of  $T(G)$  such that:

There exists a permutation  $\sigma$  of  $\{1, \dots, 2^t\}$  such that:

$$\begin{aligned} T_{lk} &= T(G)_{\sigma(l)k} \oplus csum(k, T), & \text{if } k < t+1, \\ T_{lk} &= T(G)_{\sigma(l)k} \oplus csum(k, T) \oplus rsum(l, T), & \text{if } k = t+1, \end{aligned}$$

where  $T_{lk}$  (resp.  $T(G)_{lk}$ ) is the entry in the  $l$ 'th row and  $k$ 'th column of  $T$  (resp.  $T(G)$ )  $\square$

## 8.4. The Protocol.

In the protocol we will need a source of randomness that all participants can trust. We therefore start with a description of a sub protocol that can be used to simulate such a source. All references in the following to “the random source” will be taken to mean executions of this sub protocol. It assumes that all participants have chosen an instance of a bit commitment scheme based on claw free functions. Note, however, that the instance used in the sub protocol is NOT necessarily the same as the one used in the main protocol below.

### PROTOCOL COIN FLIP

1. For  $i = 1 \cdots n$ ,  $P_i$  chooses at random a bit  $b_i$  and makes a commitment to  $b_i$  public.
2. For  $i = 1 \cdots n$ ,  $P_i$  opens his commitment to  $b_i$ .
3. The result of the coin flip is computed as  $b_1 \oplus \cdots \oplus b_n$ .

Notice now that, because the commitment schemes used hide the bits unconditionally, the distribution of each  $b_i$  is independent of all other bits chosen, assuming that no participant can change his mind about his choice. Under this assumption, the result will be a uniformly chosen bit, if at least one participant executes Step 1. correctly. Thus we have

#### Lemma 8.1

Let  $r_1, \dots, r_{P(m)}$  be a sequence of bits produced by a polynomial number  $P(m)$  of executions of the COIN FLIP protocol. Then in all but a super polynomially small fraction of the cases, the  $r$ 's are uniformly chosen, independent bits.

#### Proof.

By the Unforgeability property of the commitment schemes used  $\square$

Using a commitment scheme based on probabilistic encryption here would also work, although the proof of correctness of the main protocol would become somewhat more complicated, due to the fact that the distribution of the  $r$ 's would then not be exactly uniform, although very close. With the implementations of bit commitments currently known, however, there is no need to introduce this complication: we do need the blinding property of bit commitments for the main protocol, and the only known way to get this is by assuming QRA, which is certainly a stronger assumption than IAF, which in turn is sufficient to construct claw free functions, as we have seen.

The first step in the main protocol is that each participant chooses an instance of a bit commitment scheme and uses the protocols mentioned in Section 5 to convince all other participants that the choice was made correctly. We assume for simplicity that all participants use the same value of  $m$  as security parameter, but in principle, different participants could choose different levels of security.



The commitment scheme chosen by participant  $P_i$  is called  $BC_i$ .

All commitment schemes  $BC_i$  with  $i < n$  must satisfy all properties defined in Section 5, so they can be instances of QRS, but not of JSS or DLS.  $BC_n$  must satisfy all the properties from Section 5, except the blinding property, so it could be an instance of JSS or DLS.

The main part of the protocol proceeds in two phases: 1) The Encryption Phase and 2) The Computation Phase. Each is described formally in separate subsections below.

#### 8.4.1. The Encryption Phase.

The procedure TRANSFORM CIRCUIT below is iterated once by each participant, such that the  $i$ 'th iteration is executed by  $P_i$ . After each iteration the protocol CHECK TRANSFORMATION is executed to verify that the preceding iteration of TRANSFORM CIRCUIT has been performed correctly. The input to the first iteration will be the truthables from the original circuit.

##### PROCEDURE TRANSFORM CIRCUIT.

*Input: in the  $i$ 'th iteration, one  $(i-1)$ -transform of each truthable in  $C$ . Output: one  $i$ -transform of each truthable in  $C$ .*

For each  $(i-1)$ -transform  $S$  supplied in the input, do the following:

- 1) Suppose the corresponding gate in  $C$  has  $t$  input wires, so that  $S$  has  $2^t$  rows and  $t+1$  columns. Choose a permutation  $\sigma$  of  $\{1, \dots, 2^t\}$  at random and apply  $\sigma$  to the rows of  $S$  with attached lists. Call the result  $T$ .
- 2) For  $l=1 \dots 2^t$ , and each  $j < i$ , do:  
choose a bit  $s_{ij}(l, T) \in \{0,1\}$  at random and x-or the last entry in the  $l$ 'th row of  $T$  with  $s_{ij}(l, T)$ . Find the commitment

$$BC_j(b_j(l, T), r_j(l, T)) \in rlist(l, T),$$

and use the blinding property of  $BC_j$  to replace this commitment with

$$BC_j(b_j(l, T) \oplus s_{ij}(l, T), r'_j(l, T)) \in rlist(l, T).$$

- 3) For  $k=1 \dots t+1$  do:

if the  $k$ th column in  $T$  corresponds to an input wire which is not in  $I_i$ , then do nothing.

otherwise, choose a bit  $b_i(k, T) \in \{0,1\}$  in the following way: If column  $k$  in  $T$  is connected to another column in another transform for which a bit  $b$  has already been chosen, then put  $b_i(k, T)=b$ . Otherwise, choose  $b_i(k, T)$  at random. Now x-or all entries in the  $k$ 'th column of  $T$  with  $b_i(k, T)$ , and append to  $clist(k, T)$  a commitment

$$BC_i(b_i(k, T), r_i(k, T)).$$

4) For  $l=1 \cdots 2^t$  do:

choose a bit  $b_i(l,T) \in \{0,1\}$  at random, x-or the last entry in the  $l$ 'th row of  $T$  with  $b_i(l,T)$ , and append to  $rlist(l,T)$  a commitment

$$BC_i(b_i(l,T), r_i(l,T)).$$

Notice that the special way of choosing  $b_i(k,T)$  in step 3 ensures that whenever two columns are connected, the contents of the corresponding column lists will be identical.

The purpose of step 2 is to hide the row permutation  $\sigma$ . The blinding in this step ensures that the contents of the row lists in  $S$  and  $T$  are statistically independent, so that no information about  $\sigma$  is revealed by the row lists. The reader may have noticed that information about  $\sigma$  is revealed by the contents of the input columns in  $T$ . But since each such column contains as many 0's as 1's, a particular entry in a column still has the same probability of being the encryption of a 1 as of a 0, if the bit x-ored into the column by  $P_i$  is unknown. This will be sufficient to make the protocol secure, as we shall see.

The interactive proof that every  $T$  was created according to the protocol proceeds as follows:

#### PROTOCOL CHECK TRANSFORMATION

*Input:* For every truth table in  $C$ , an  $(i-1)$ -transform  $S$  and an  $i$ -transform  $T$ , which  $P_i$  claims was created correctly from  $S$ .

- 1) For every corresponding pair  $(S,T)$  in the input,  $P_i$  creates from  $S$  another  $i$ -transform  $T'$  in exactly the same way as  $T$ , but with new independent choices of permutation and bits.  $T'$  is made public.
- 2) The source of random bits chooses  $b \in \{0,1\}$  at random.
- 3) If  $b=1$ ,  $P_i$  must for every  $T'$ : make public the row permutation used for creating  $T'$ , open all his bit commitments attached to  $T'$ , and reveal all blinding factors (see Section 5) he used in blinding other commitments. This allows everybody to check that  $T'$  was correctly constructed.
- 4) If  $b=0$ ,  $P_i$  must show for all corresponding  $(T,T')$  a relation between  $T$  and  $T'$ :

$P_i$  makes public the permutation  $\pi=\sigma(\sigma')^{-1}$ , where  $\sigma$  (resp.  $\sigma'$ ) is the permutation used in creating  $T$  (resp.  $T'$ ).

For  $k=1 \cdots t+1$ : if  $P_i$  appended commitments to  $clist(k,T)$  and  $clist(k,T')$ ,  $P_i$  uses comparability of  $BC_i$  to show from his commitments the value of

$$b_i(k,T) \oplus b_i(k,T').$$

For  $l=1 \cdots 2^t$ ,  $P_i$  uses comparability of his commitments in  $rlist(l, T)$  and  $rlist(\pi(l), T')$  to show the value of

$$b_i(l, T) \oplus b_i(\pi(l), T').$$

For  $l=1 \cdots 2^t$ , and each  $j < i$ ,  $P_i$  uses the blinding property of  $BC_j$  to show the value of

$$s_{ij}(l, T) \oplus s_{ij}(\pi(l), T').$$

This allows everybody to check that the following holds:

$$T_{lk} = T'_{\pi(l)k} \oplus csum(k, T) \oplus csum(k, T'), \quad \text{if } k < t+1,$$

$$T_{lk} = T'_{\pi(l)k} \oplus csum(k, T) \oplus csum(k, T') \oplus rsum(l, T) \oplus rsum(\pi(l), T'), \quad \text{if } k = t+1.$$

Steps 1 - 4 are repeated  $m$  times.

### Theorem 8.2.

For every  $i=1 \cdots n$ , the probability that  $P_i$  has cheated (i.e. sent incorrect messages) anywhere in the protocol without this being detected is smaller than  $m^{-c}$  for any constant  $c$  and all sufficiently large  $m$ . If no cheating occurs, then the encryption phase ends with  $P_n$  outputting a valid  $n$ -transform of every truth table in  $C$ .

#### Proof.

It is not hard to see that if for at least one iteration in the CHECK TRANSFORMATION protocol,  $P_i$  was able to give satisfying answers in *both* steps 3 and 4, and if  $P_i$  was not able to change the contents of his bit commitments, then  $T$  must have been correctly constructed. This, Lemma 8.1 and the unforgeability property of all bit commitment schemes used suffices to prove the first statement. The second follows from the obvious fact that if  $T$  was created correctly from  $S$  in the TRANSFORM CIRCUIT protocol, then  $T$  is valid if  $S$  was valid  $\square$

Intuitively, the CHECK TRANSFORMATION protocol is secure from  $P_i$ 's point of view because all the other participants never see anything but random "copies" of  $S$  or  $T$ , which they could also have produced themselves. All other participants will therefore be convinced that  $T$  was correctly constructed, but will learn nothing more about  $T$ . A formal proof can be found in Section 8.5.

### 8.4.2. The Computation Phase.

At the end of the encryption phase,  $P_n$  has output a set of  $n$ -transforms of the truth tables in  $C$ . This set will be called  $C^*$ .

Let  $w$  be an input wire in  $C$  connecting into the gate  $G$ , and suppose  $w$  corresponds to column  $k$  in  $T(G)$ . Let  $T^*$  be the  $n$ -transform of  $T(G)$  contained in  $C^*$ .

As the first step in the computation phase, *encrypted* input bits are specified for each such  $w$ . This is done as follows:

if  $w \in R$ , then the source of random bits selects a bit  $b_w$ , which is made public. Note that  $b_w$  should be thought of as an encrypted input bit, i.e. the actual computation will use as input from this wire the xor of  $b_w$  and secret bits chosen for the wire by the participants.

if  $w \in I_i$  for some  $i$ ,  $P_i$  reads a bit  $b$  in  $ip_i$  corresponding to  $w$  from its private input tape, and then makes  $b_w = b \oplus b_i(k, T^*)$  public.

To describe the next steps in the computation phase, we need to define the notion of computations in  $C^*$ .

#### Definition 8.4.

By a *computation*  $\Gamma^*$  in  $C^*$ , we mean a selection of exactly one row in each  $n$ -transform in  $C^*$ .

A computation  $\Gamma^*$  is said to be *consistent*, if the following is satisfied:

Suppose output column  $k_1$  in the transform  $T_1^*$  is connected with column  $k_2$  in the transform  $T_2^*$ , and let  $\Gamma^*(k_1)$  and  $\Gamma^*(k_2)$  be the entries selected by  $\Gamma^*$  in the two columns. Then

$$\Gamma^*(k_1) \oplus rsum(l_1, T_1) = \Gamma^*(k_2),$$

where  $l_1$  is the index of the row selected by  $\Gamma^*$  in  $T_1^*$   $\square$

The intuition behind these rather technical looking definitions is very simple: if we assume that all transforms in  $C^*$  are valid, then the selection of a row in a table  $T^*$  corresponds to selecting a row in the original truth table, under the product of all row permutations chosen for  $T^*$  by the participants. Therefore, a computation in  $C^*$  corresponds to a computation in  $C$ . Moreover, the consistency condition on a computation in  $C^*$  implies that if we look at the string of bits selected in the computation and “remove” all layers of encryption, then the consistency condition as defined in Section 8.3 holds for the computation in  $C$  (note that by construction of the  $n$ -transforms in  $C^*$ ,  $csum(k_1, T_1^*) = csum(k_2, T_2^*)$ ). Formally, we have the following:

Given an  $n$ -transform  $T^*$  in  $C^*$ , let  $\sigma_i$  be the row permutation chosen by participant  $P_i$  for  $T^*$ . If  $\Gamma^*$  is a computation in  $C^*$  selecting row  $l$  in  $T^*$ , then we define the *corresponding computation*  $\Gamma$  in  $C$  by selecting row  $(\sigma_n \cdots \sigma_1)^{-1}(l)$  in the original truth table.

#### Lemma 8.3

Let  $\Gamma^*$  be a computation in  $C^*$  and suppose all transforms in  $C^*$  are valid. Then the corresponding computation  $\Gamma$  in  $C$  is consistent if and only if  $\Gamma^*$  is consistent  $\square$

The rest of the protocol is now just an algorithm which constructs a consistent computation in  $C^*$  from the input as specified above.

#### PROCEDURE CONSTRUCT COMPUTATION

*Input: The fully encrypted circuit  $C^*$  and for each input wire  $w$  an attached input bit  $b_w$  specified as above. Output: A consistent computation  $\Gamma^*$  in  $C^*$  and its output.*

- 1) Mark every input wire.
- 2) For every gate  $G$  for which all input wires are marked, do the following:  
Let  $T^*$  be the  $n$ -transform in  $C^*$  of  $T(G)$ . Let  $l$  be the index of the row in  $T^*$ , whose first entries match the bits attached to the input wires of  $G$ , and let  $b_l$  be the last entry in this row.

Record row number  $l$  as selected by  $\Gamma^*$ . Open all bit commitments in  $rlist(l, T^*)$ , and put

$$b_l^* = b_l \oplus rsum(l, T^*).$$

- 3) For every wire  $w$  connecting out of  $G$ , mark  $w$  and attach to it

$$b_w = b_l^*.$$

- 4) If any wires in  $C$  are still unmarked, go to Step 2).
- 5) For every output wire  $w$ , do the following:  
suppose  $w$  corresponds to output column  $k$  in  $T^*$ . Then open all commitments in  $clist(k, T^*)$ , and compute the final result for this wire as:

$$Result(w) = b_w \oplus csum(k, T^*).$$

The reader can easily verify that if all participants supply the information needed in step 2, then the procedure runs in time linear in the number of gates in  $C$  (assuming that the fan-in is bounded by a constant), and constructs a consistent computation  $\Gamma^*$ . Furthermore, it is easy to see that step 5 above in fact produces the output of the computation in  $C$  corresponding to  $\Gamma^*$ .

### 8.5. Proofs of Correctness and Minimum Knowledge.

Before starting on this section, the reader is well advised to review the definitions of correctness and minimum knowledge given in Section 6. Also the notation from that section will be used in the following.

Using the results proved in preceding sections, it is now not hard to prove



### Theorem 8.4

Under IAF, the protocol is correct.

#### Proof

First note that for each input wire  $w$  in  $R$  (the set of random input wires), the input bit given to  $C$  is the x-or sum of one bit chosen by the random source, and one bit chosen by each participant (namely the bit committed to in the *clist* corresponding to  $w$ ). By Lemma 8.3, this means that the conditional probability distribution of the output of the protocol assuming that no cheating has occurred and that the COIN FLIP protocol has produced correctly distributed bits, is exactly equal to the “right” probability distribution  $OP_I$ , for any choice  $I$  of inputs. The theorem now follows from Lemma 8.1, Theorem 8.2 and elementary probability theory  $\square$

To show that the protocol is minimum knowledge, we assume the existence of some conspiracy  $X$ . We must then exhibit a simulator  $M_X$  for  $X$  and prove that its output is polynomially indistinguishable from an actual protocol conversation. Only informal descriptions will be given, we trust that this will make the proofs easier to understand, and that the reader will have no trouble in filling in the necessary details.

We begin with the description of the simulator  $M_X$ :

Recall that  $M_X$  is given as input the security parameter value  $m$ , the output of a “real” computation  $f(ip_1, \dots, ip_n, ra)$ , and the part of the input “known” to  $X$ ,  $\{ip_i \mid P_i \in X\}$ . In addition to this,  $M_X$  is of course allowed to use the machines in  $X$  in any (feasible) way it likes. We will describe the algorithm of  $M_X$  as a simulated protocol execution, where the participants in  $X$  act “as themselves” and  $M_X$  plays the parts of all other participants.

$M_X$  starts by putting  $ip_i$  on the input tape of  $P_i$  for every  $P_i \in X$ . Also, the random tapes of all machines in  $X$  are filled in using  $M_X$ ’s own random tape. This means that all the machines in  $X$  are “deterministic” from now on.

For simplicity, we assume that each participant uses QRS for bit commitments.  $M_X$  therefore chooses a modulus  $N_i$  for all participants  $P_i$  not in the conspiracy, while the participants in  $X$  choose their own moduli. Although this means that  $M_X$  in fact knows the factorizations for the honest participants, it is of course essential that this is not used in the simulation!

Next,  $M_X$  executes a simulation of the proof that  $N_i$  is a Blum integer, for every  $i=1 \dots n$ . When  $i$  is such that  $P_i \in X$ ,  $P_i$  can be used directly as the prover, otherwise the simulation from [GrPe] can be used.

We now come to the executions of the TRANSFORM CIRCUIT and the CHECK TRANSFORMATION procedure. If  $P_i \in X$ , then the  $i$ ’th iteration of this procedure is replaced by an execution of the procedure SIMULATE CONSPIRACY below, otherwise the procedure SIMULATE HONEST PARTICIPANT is used.

### PROCEDURE SIMULATE HONEST PARTICIPANT.

*Input: one (i-1)-transform of every truth-table in  $C$ . Output: one i-transform of every truth-table in  $C$ .*

- i) For every (i-1)-transform  $S$  supplied in the input, which does not correspond to an output gate,  $M_X$  creates an i-transform  $T$  exactly according to the protocol.
- ii) For every (i-1)-transform  $S$  corresponding to an output gate, an i-transform  $T$  is created in exactly the same way as in step i. In addition to this, the following is done:

Recall that for each output gate  $G$  of  $C$ , a bit  $b_G$  is specified in the input to  $M_X$ , as part of the given value of  $f(ip_1, \dots, ip_n, ra)$ . The simulated computation must produce this bit as output from  $G$ . This is ensured by simply multiplying the commitment from  $P_i$  in some of the row lists by  $-1$ , so that the last column of  $T$  becomes an encryption of a column with all entries equal to  $b_G$ .

Note that the above modification has to be done once only, even if the simulation is done with more than one honest participant.

- iii) To simulate the CHECK TRANSFORMATION protocol,  $M_X$  chooses  $b_M \in \{0,1\}$  at random. If  $b_M=1$ , a set of  $T''$ 's is created as valid transforms from the  $S$ 's, otherwise, the  $T''$ 's are created from the  $T$ 's as randomly chosen i-transforms such that step 4 in the CHECK TRANSFORMATION protocol can be executed.
- iv) The trusted source outputs a bit  $b$ . If  $b=b_M$ ,  $M_X$  just executes step 3 in the CHECK TRANSFORMATION protocol if  $b=1$ , and step 4 if  $b=0$ . This is possible by construction of  $T'$ . Otherwise,  $M_X$  rewinds all machines in  $X$  to their configuration just after the last execution of step ii. We then go back to step iii and try again with a new independent choice of  $b_M$  and  $T'$ .

Steps iii and iv are repeated until step iv has been successful  $m$  times.

Note that the expected number of "trial  $T$ 's", we must create in the above procedure before  $b$  happens to be equal to  $b_M$  is 2, and that the above procedure therefore takes expected linear time in  $m$ .

### PROCEDURE SIMULATE CONSPIRACY.

*Input: one (i-1)-transform of every truth-table in  $C$ . Output: one i-transform of every truth-table in  $C$ .*

- i)  $M_X$  uses  $P_i$  to compute an i-transform  $T$  of every (i-1)-transform  $S$  in the input.
- ii) The first round of the CHECK TRANSFORMATION protocol is executed as follows: step 1 is executed only once, but steps 2-4 are executed several times, rewinding  $P_i$  after each iteration. This goes on until  $P_i$  has shown *both* that  $T$  is equivalent to  $T'$ , and that  $T'$  was correctly constructed. If either proof is not valid, the simulator stops.

The reader can verify that having seen both proofs,  $M_X$  can find out exactly how  $T$  was constructed from  $S$ . In particular, all the blinding factors used by  $P_i$  (and their square roots) can be found. This means that  $M_X$  can now open all bit commitments used so far in the simulation, without making further use of the machines in the conspiracy.

- iii) The rest of the rounds in the CHECK TRANSFORMATION protocol are executed exactly as in an actual protocol execution.

Once again, the above procedure takes expected polynomial time.

In the computation phase,  $M_X$  lets all the machines in  $X$  specify their encrypted choice of input bits, and specifies randomly chosen bits for all other participants. It is now easy to see that the CONSTRUCT COMPUTATION procedure can be executed exactly as in the protocol: By the remarks in the SIMULATE CONSPIRACY procedure,  $M_X$  has the information it needs to open all bit commitments from all participants, because the square roots of all blinding factors are known to  $M_X$  at this point.

Why does this simulation look just like a “real” protocol conversation? Consider the set of messages sent by participant  $P_i$  in the protocol, and suppose  $P_i$  is not in  $X$ . It is easy to see that the bits shown “in clear” in these messages will be distributed in exactly the same way in the simulation as in an actual protocol conversation. The difference between simulation and protocol therefore lies only in the distribution of the bits hidden in  $P_i$ ’s bit commitments. It is intuitively clear that to notice this difference, a distinguisher would need the ability to tell the difference between commitments containing 0’s and those containing 1’s. But by assumption, this cannot be done efficiently. This argument is formalized below.

### Theorem 8.5

Under QRA, the protocol is minimum knowledge.

#### Proof.

It remains to be shown that the output of the simulator described above is polynomially indistinguishable from an actual protocol conversation. For simplicity, we will only do this in the case where  $n-1$  participants conspire against one honest participant  $P_i$ . Cases with smaller conspiracies are in principle similar and introduce only technical differences.

Recall that a polynomial time distinguisher  $\Delta$  is a probabilistic polynomial time algorithm which takes as input a binary string and gives a one bit output. Let  $p_{prot}(m)$  be the probability that  $\Delta$  outputs a 1 when given an actual protocol conversation with security parameter  $m$  as input. Similarly,  $p_{sim}(m)$  is the probability that  $\Delta$  outputs a 1 when given a simulated conversation created as above. By way of contradiction, we assume that there exists a choice of inputs  $(ip_1, \dots, ip_n, ra)$  which will result in distinguishable protocol and simulated conversations, i.e. there exists a constant  $c$ , such that

$$|p_{\text{prot}}(m) - p_{\text{sim}}(m)| > m^{-c} \quad (1)$$

for infinitely many  $m$ .

Below, we will derive a contradiction with the quadratic residuosity assumption by showing how to use  $\Delta$  to construct an algorithm that will distinguish quadratic residues from non residues modulo a Blum integer  $N$ , whenever  $N$  has  $m$ -bit prime factors, and  $m$  satisfies the above equation. Let us therefore assume that we are given an element  $x$  in  $\mathbb{Z}_N^*$  of Jacobi symbol 1.

Below, we will describe a polynomial time Turing machine  $M_X'$ , which on input  $(ip_1, \dots, ip_n, ra)$  and  $x$  chosen as above will generate a conversation. With overwhelming probability, this conversation will be a “real” protocol conversation if  $x$  is a quadratic residue, and it will be distributed exactly as a simulated one if  $x$  is a quadratic nonresidue.

The intuitive idea behind the construction of  $M_X'$  is the following: By assumption, we are given a complete set of secret inputs. It is not surprising that from this, one can generate a genuine protocol conversation using these inputs (some rewinding of participants in  $X$  may be necessary, since the factorization of  $N$  is not known). But as mentioned before, the only difference between protocol and simulation lies the distribution of bits hidden in commitments, and therefore, the genuine conversation can be turned into a simulated one by complementing some of the hidden bits. So if we multiply the corresponding commitments by  $x$ , the conversation will be a simulated one, precisely if  $x$  is a quadratic non residue.

The description of  $M_X'$  is based on the algorithm of  $M_X$ :  $M_X'$  assigns  $N$  as modulus to  $P_i$ , and then works exactly as  $M_X$ , except for steps i and ii in the SIMULATE HONEST PARTICIPANT procedure. In step i,  $M_X'$  will: for each commitment placed by  $P_i$  in an input column list choose a bit  $b$  at random and multiply the commitment by  $x^b$ . To see why this is done, observe that in a real protocol conversation the bits used in the (encrypted) computation and the bits contained in column lists are correlated: they always x-or together to the bits used in the unencrypted computation. If  $x$  is a quadratic nonresidue, this correlation is destroyed by the multiplications by  $x$ , corresponding to the fact that we should be producing a simulated conversation in this case. The resulting transformation, however, is no longer valid. Since both simulation and protocol conversations produce valid transforms for non output gates, we have to do something about this. But the correctness of the transform can be restored by also multiplying appropriately chosen commitments in row lists and output column list by  $x$ . Note that  $M_X'$  will know how to do this, since it knows exactly how every transform was computed. Finally in step ii, the multiplications by  $-1$  are replaced by multiplications by  $x$ .

In the computation phase,  $M_X'$  specifies inputs for  $P_i$  according to the protocol, i.e as if the  $x$ 's had not been multiplied in. The reader may have observed that a problem could arise here: To complete this phase,  $M_X'$  must be able to open some of the commitments from  $P_i$  placed in row lists. But this is of course impossible if the commitment has been multiplied by  $x$ . Note, however, that since all inputs are known to  $M_X'$ , it is known which rows will be used in the computation in  $C$ , and therefore it can be predicted which rows will be used in the



encrypted computation. This, together with the observation that the necessary adjustments above can always be done while leaving at least one row untouched, solves the problem.

Now recall that if  $x$  is a quadratic residue, multiplying a commitment by  $x$  does not change the bit contained in the commitment, while if  $x$  is quadratic nonresidue, the bit is complemented. Using this fact, we can verify that the claim above about the output of  $M_X'$  holds: the distributions match exactly, except in the case where a dishonest participant transforms the circuit incorrectly, and is not caught. In this case  $M_X'$  has to stop, while the real protocol will compute some (incorrect) result. But this possibility only occurs with negligible probability, by Theorem 8.2.

Let  $p_{prot}(m | N)$  be the probability that  $\Delta$  outputs a 1 when given as input a protocol conversation in which  $P_i$  chooses  $N$  as modulus.  $p_{sim}(m | N)$  is defined similarly. An easy calculation will show that (1) implies that there exists  $c'$ , such that:

$$|p_{prot}(m | N) - p_{sim}(m | N)| > m^{-c'} \quad (2)$$

for infinitely many  $m$  and for more than a negligible (i.e. polynomial) fraction of the possible choices of  $N$ . Elementary probability theory now shows that if  $x$  is a randomly chosen element in  $\mathbb{Z}_N^*$  of Jacobi symbol 1, then we can guess whether  $x$  is a quadratic residue with an advantage polynomially larger than 0.5, whenever (2) holds. But this contradicts the hiding property of QRS, i.e. the intractability assumption on the quadratic residuosity problem  $\square$

### Theorem 8.6.

If  $P_n$  is honest and uses DLS for bit commitments, then the protocol releases no information in the Shannon sense about the private input of  $P_n$  and all intermediate results in the computation.

**Proof.**

Since DLS hides the bits unconditionally, it is easy to see that the protocol provides even a conspiracy consisting of all other participants with nothing more than the string of input bits and intermediate results, encrypted under a true one-time pad  $\square$

A similar result holds if  $P_n$  uses JSS, *assuming* that the moduli used are of the right form. Since the known interactive proofs of this would leave a small probability that this is not so,  $P_n$  would on the average be releasing an exponentially small amount of information when using JSS.

## 8.6. Generalizations.

In this section, we will show how to adopt the protocol to various additional requirements. It will be shown that a very flexible functionality can be obtained from a general computation protocol, as long as it protects the private inputs, computes correct results, and allows execution of *coordinated* instances, i.e. participants can prove relations between inputs and



outputs used in different instances of the protocol. In particular, a protocol which implements the “input-secure computation”-primitive as defined in [GaHaYu] and allows coordinated instances can in fact implement all four “primitives for cryptographic computation” defined in [GaHaYu]. From the comparability property of the bit commitment schemes used, it is clear that our basic protocol satisfies these requirements.

### 8.6.1. Private Outputs.

How can we make some of the output bits private to a participant  $P_i$ ? First, note that this problem could always be solved by rewriting the circuit, such that each of the output bits in question were x-ored with a bit chosen at random by the participant.

With our protocol, however, this is not needed: we can just modify the protocol such that  $P_i$  is not required to open his bit commitments corresponding to the output wires in question. The proof of security could easily be modified to take this into account: just note that if some output bits are private to a participant not belonging to a conspiracy, then the simulation can be done without knowing these bits (here we assume that no participant will stop the protocol too early - if this is not satisfied, the countermeasures from the next section can be applied).

### 8.6.2. Gradual Release of the Output.

An obvious strategy for a set of dishonest participants would be to try stopping the protocol early in such a way that this conspiracy could find out about the output while preventing honest participants from getting it. To solve this problem in general, one could just execute a set of coordinated instances, such that the input was constant over all rounds, while the circuit was rewritten such that exactly one bit of the result was computed in each instance.

Note, however, that our basic protocol already implements a bit by bit release, since the “encryptions” of the output columns are opened for one column at a time. Thus, each participant will never have more than a one bit advantage over any other participant. Also this statement could be incorporated into the formal security proof: loosely speaking, if the protocol stops without computing the complete result, then the simulation can be done without knowing all output bits.

If simultaneous release is to be combined with private outputs, however, then a bit by bit release does not make sense because the private outputs may have different lengths. We propose instead the following solution, the basic idea of which was first introduced by Yao in [Ya]: The parties agree on a probabilistic circuit, which will produce as output an instance of a bit commitment scheme based on a trapdoor one way function, e.g an instance of QRS, together with the trapdoor information. Our basic protocol can now be used to do a computation in this circuit. The parties open the specification of the instance, such that commitments can be computed, but keep the trapdoor closed. In the case of QRS, this would mean that a public modulus has been computed, but its factorization is still unknown to everyone. Since all coinflips in the computation are secret, no participant can compute the trapdoor information by himself. The parties now go back to the original circuit and do the computation, using

the bit commitment instance computed above for all commitments in output column lists. When this is done, the trapdoor information is opened, bit by bit, which is possible by the above remarks. Using Yao's terminology, this is a *fair* protocol because all participants need exactly the same information in order to get their share of the output.

### 8.6.3. Other Special Requirements.

Since our basic protocol protects not only the inputs, but also all intermediate results, a variety of special properties can be obtained by rewriting the circuit. For example, the result could be distributed only to a secret subset of users, the subset being chosen at random or based on the result. Also, a secret permutation could be applied to the private output of some participant to hide the order in which he obtains his output bits. This could be important, e.g. in the implementation of a game such as poker.

### 8.6.4. Verifiability.

Is it possible for a non participant to check from, say, a recording of all messages sent in the protocol that the result computed is a correct one? It is clear that such a check cannot be possible without interacting with the participants, since otherwise the protocol could not be minimum knowledge with respect to the private inputs: recall that there should be no way for anyone to tell whether he is presented with a genuine protocol conversation or a simulated one. But this is exactly what we would be asking the non participant above to do!

With interaction, however, checking is easy: each participant could be asked to prove once again using the CHECK TRANSFORMATION protocol that he has computed valid transforms. Then all of the computation phase could be checked without interaction.

A variation on this line of thought leads to the construction of a credentials mechanism:

### 8.6.5. Fault Recovery.

In our basic construction, there is not much one can do short of stopping the protocol, if some participant sends incorrect messages, or just stops completely. In the recent literature, a number of techniques have been proposed for recovering from such situations.

Under the assumption that the majority of users are honest, [GoMiWi] propose to verifiably secret share the secret inputs of all participants. In the event of a fault, a majority of users would recover all the secrets of the participant who stopped, and complete the computation. Thus, it is assumed by definition that an honest participant will always send the messages he is supposed to. This hardly seems a realistic assumption, however: in practice, almost all faults may occur by accident, or even worse, as a result of dishonest participants sabotaging honest ones to reveal their secrets!

Galil, Haber and Yung [GaHaYu] propose instead a secret sharing in two levels: first, for each secret bit of a user, the user distributes one "x-or share" to each of the other participants, where the exclusive-or of these x-or shares is the original secret bit. The x-or shares are then verifiably secret shared. Recovery is now possible by reconstructing the x-or shares held by

the stopped participant, which has the effect of preserving that participant's privacy.

Our protocol could easily be adapted to include the procedure of [GaHaYu]: In case QRS is used, participant  $P_i$  would first commit to his input, and then distribute x-or shares of the factorization of his modulus to the other participants. If  $P_i$  stops working, any majority of users could reconstruct the x-or shares of his factorization, and use instances of our basic protocol to simulate  $P_i$ : open his commitments as needed, etc.

In addition to this, the techniques presented in [Ch2] can be used to add flexibility to the protocol. there, it is shown how to change the *quorum*, i.e. the number of participants necessary to reconstruct the secret, during the protocol. This can be useful in applications, where it can be assumed that almost all faults occur by accident, because of breakdown of communication lines, etc. In this case, it may be desirable to keep the ratio between the quorum and the number of remaining participants constant.

#### 8.6.6. The Possibility of Further Generalizations.

From an intuitive point of view, it may seem unnatural that only one participant can be unconditionally protected. Why not try to devise a protocol in which any subset of the participants could have this option?

To understand this question better, consider the All-or-nothing-disclosure problem (ANDOS). This problem was presented and studied in [BrCrRo].

In this problem, party  $A$  possesses a number of secrets, of which she is willing to disclose exactly one to  $B$ .  $B$  would like to be able to choose which secret to get, without disclosing to  $A$  which secret he is interested in.

Clearly, ANDOS is just a special case of the general computation problem. Furthermore by an argument by Claude Crépeau, fundamental information theoretical reasons imply that a two party ANDOS cannot be implemented such that the secrets of both parties are unconditionally protected. Informally, this is so since if  $A$  is unconditionally protected, the messages she sends must contain enough Shannon information to determine exactly one of her secrets, and nothing about the other one. But this must then be the secret  $B$  learns, which means that he is anything but unconditionally protected!

Although we do not yet have a formal proof that this last claim is true, it certainly seems that there is little hope of achieving the generalization described above.

Under different assumptions, however, it *is* possible to achieve unconditional protection for all participants. This is described in more detail in the next section.

On the positive side, it should be noted that in joint work with Claude Crépeau, we have shown that our basic protocol can be made to work, based only on the assumption that a one way function exists, and that there is a protocol solving the ANDOS problem: Clearly, bit commitments are possible if one way functions exist, and the comparability property can always be satisfied using the fact that all NP-statements can be proved in minimum knowledge (although using very inefficient protocols). The idea is now to use a protocol for ANDOS to

get something which will replace the blinding property: Notice that when participant  $P_i$  blinds other participants' commitments, the basic fact used in the protocol is that  $P_i$  knows the x-or sum of the bit contained in the original commitment, and the one contained in the blinded version. It will therefore be sufficient if he can learn a set of these x-or relations by doing an ANDOS protocol with each  $P_j$  with  $j < i$ .

The fact that ANDOS can be based on the existence of a trapdoor one way function [Cr] then implies that our protocol can also be based only on a trapdoor one way function. The resulting protocol would have the same property as the one of [GaHaYu] which is based on the same assumption, namely that only one trapdoor function would have to be generated for each participant, while the solution in [GoMiWi] requires generation of  $O(n^2)$  functions.

## 8.7. Applications.

In this subsection, we will briefly discuss two applications of our protocol which are of special interest, since they make essential use of the fact that one participant can be unconditionally protected.

### 8.7.1. Credentials.

In the following, we describe briefly a new construction for *credential mechanisms*, a concept invented David Chaum. In short a credential mechanism is a protocol, or rather a system of protocols for transmitting personal information between organizations. This information takes the form of *credentials*, for example driver's licenses, ID-cards, etc. – in other words, messages from the organization issuing the credential, stating that a given type of credential applies to a given individual. We would like a protocol implementing a credential mechanism to achieve two things:

- 1) No individual can show a credential to anyone, unless it has been properly issued to him.
- 2) Credentials are *untraceable* from the point of view of the organizations, i.e. even if all organizations cooperate, they will get no information allowing them to construct a complete image of any individual: which credentials apply to him, when and where he has shown them, etc.

While individuals in general have limited computing resources, large organizations may have much larger, and often unknown resources. It therefore makes good sense to require that 1) is satisfied relative to some intractability assumption, while 2) is unconditionally satisfied.

In [ChEv], Chaum and Evertse present a credential mechanism, based on the security of RSA. This construction is quite practical, but its security can only be proved in a restricted formal model, and one must assume the existence of a center which has ability to compute all types of credentials, and must be able to do so on line. It must therefore be trusted by the organizations, but not by the individuals.

In the following, we present a construction, which is far from being practical, but has a number of theoretical advantages: it is the first construction whose security can be proved



relative to a single plausible intractability assumption, and can ensure that only organizations authorized to issue a given credential type are able to compute these credentials. We do need a center that the organizations must trust, but it is only used to validate each individual in the system. This is necessary only once for each individual, and this center can therefore work off-line.

Assume that some unique piece of information  $ID(i)$  is assigned to individual  $i$  (like name, address, etc.). One then fixes the rule that a given type of credential applies to  $i$  if he possesses a signature in some signature scheme on  $ID(i)$ . Each organization authorized to issue this credential type is therefore assumed to know the secret key of the corresponding signature scheme.

The solution is now simply that instead of giving  $ID(i)$  to all organizations,  $i$  will give commitments to the bits in  $ID(i)$ , computed in an unconditionally secure bit commitment scheme, and with new randomly chosen commitments for each organization. These commitments act as the *pseudonyms* in Chaum's and Evertse's construction. When organization  $O$  issues a credential to  $i$ , we use the basic computation protocol to compute the relevant signature on  $ID(i)$ , such that  $O$  enters the secret key for the signature scheme as private input, while  $i$  enters  $ID(i)$ . The signature is then computed as private output for  $i$ . When  $i$  later wants to show his credential to another organization, he can use a separate instance of our protocol to convince the second organization that he really knows the required signature.

Since the computation protocol is minimum knowledge w.r.t.  $i$ , no credential can be shown if it has not been issued, assuming that the signature scheme itself is secure against chosen message attacks. And since individuals are unconditionally protected in all interactions with organizations, even with infinite computing power, the organizations cannot link the information they have on individuals.

One difficulty remains: so far nothing prevents different pseudonyms used with the same organization to represent the same individual. Even worse: if all pseudonyms are computed from the same  $ID$ , then any individual can show any credential ever issued! This is where we need the center ( $Z$ ) mentioned above. Its role will be to force individuals to form their pseudonyms properly. Before entering the system, each individual  $i$  must do the following: Go to  $Z$  and show his  $ID$  in clear to  $Z$ , who can then check it against  $i$ . Then for each organization  $O$ ,  $Z$  and  $i$  do a computation protocol, where  $Z$  signs a bit-string which is the concatenation of  $ID(i)$ , the identification of  $O$ , and a random bit-string  $R_O$ . The protocol is set up such that  $ID(i)$  and the identification of  $O$  are public input, while  $R_O$  is chosen in private by  $i$ .  $Z$ 's signature is given as private output to  $i$ .  $i$  will now use  $R_O$  as the random input necessary to compute his pseudonym with  $O$ . When later he starts communicating with  $O$ , he must first do another instance of the computation protocol with  $O$  on a circuit which takes as input the concatenation of  $ID(i)$ , identification of  $O$ , and  $R_O$ ; the signature from  $Z$  on this; and the pseudonym  $i$  wishes to use with  $O$ . It will be obvious from the above which of the inputs must be private. The circuit then checks  $Z$ 's signature, and checks that use of  $R_O$  does indeed lead to the pseudonym  $i$  wants to use. After this,  $O$  can be convinced that it is really talking



to an “existing” person, different from anyone else it has had contact with.

By the remarks in the previous section, our computation protocol can be executed, if trapdoor one way permutations exist, and one party can be unconditionally protected, if claw free permutations exist. Thus, if claw free trapdoor permutations exist, we can do the computations outlined above such that credentials will be unconditionally untraceable. The existence of claw free trapdoor permutations is also sufficient to construct a signature scheme secure against chosen message attacks, namely the GMR-scheme [GoMiRi]. Thus the whole mechanism will work relative to this condition. As we have seen, claw free trapdoor permutations exist in particular if factoring is hard, but the assumption may be more general.

### 8.7.2. Proving all IP-statements in Minimum Knowledge.

Below, we will informally discuss some methods for constructing minimum knowledge proof systems for any language  $L$  in IP. By definition,  $L$  is in IP, if an infinitely powerful Prover can interactively prove membership in  $L$  to a polynomially restricted Verifier.

In 1986, Ben-Or proved that any language in IP has a minimum knowledge proof system, if one way functions exist. Ben-Or’s proof requires use of the result by Goldwasser and Sipser on the equivalence between interactive proof systems and Arthur-Merlin games [GoSi]. Basically, the result says that anything that can be proved in protocols where the Verifier can flip coins in private, can also be proved with only public coinflips from the Verifier. Apart from being quite technical, the Goldwasser-Sipser result seems to require a prover to use infinite computing power. This is fine in the theoretical model of IP, but not very satisfying from a practical, cryptographic point of view.

The first relevant question to address therefore seems to be whether there is any difference between public and private coins when you restrict *both* the Prover and the Verifier to polynomial time computations. The following result, which was inspired by discussions with Gilles Brassard, shows that something similar to Goldwasser-Sipser holds for polynomial time provers. The complexity class MA (or NBPP) is defined exactly as NP except that the procedure for verifying a certificate of membership can be a BPP- algorithm, i.e. a probabilistic polynomial time algorithm with bounded error probability.

#### Theorem 8.7

Let  $L$  be a language in IP admitting a proof system with the following property: for each  $x \in L$ , there exists a bit-string  $w(x)$ , such that when the Prover is given  $w(x)$  as input, it can execute the proof using only polynomially bounded computing power. Then  $L \in \text{MA}$ .

#### Proof.

Let  $A = P$  or  $V$ . In each round of the protocol,  $A$  receives a message, does a polynomial time computation with inputs that could be previously sent messages, coin flips, or some trapdoor information. The output is  $A$ ’s next message in the protocol. Each of these computations can be modelled by a polynomial size probabilistic Boolean circuit. It is easy to see that these circuits can be plugged together in a way consistent with the original proof system to

obtain one large boolean circuit  $C$ , which will be of at most polynomial size. More specifically, we define the output of  $C$  to be the output from the Verifier's last computation, which is "accept" or "reject", and the input to be all coinflips made by the Prover and the Verifier, the string we want to prove membership for, and the corresponding  $w$ -value. The idea is now that if the subcircuits are connected properly, then  $C$  outputs "accept" on a set of inputs, precisely if the Verifier would have accepted in a run of the original protocol using the coinflips specified in the input to  $C$ . Now just observe that  $C$  in fact represents a BPP-algorithm for verifying membership of a string  $x$  in  $L$ , where  $w(x)$  acts as the certificate.  $\square$

Brassard, Chaum and Crépeau [BrChCr] have shown how to prove membership in any MA-language in minimum knowledge. So this together with the above shows that anything that can be proved interactively by a "practical" prover, can also be proved efficiently in minimum knowledge.

One disadvantage remains, however, as far as efficiency is concerned: the simulation in the Theorem above requires that both the Verifier's and the Prover's computations are turned into circuits. This means a huge loss of efficiency in the resulting protocol. By using the multiparty computation protocol from the previous section, we can improve this by getting rid of the circuits specifying the Prover's computations. As a side effect, we get a result that extends to all of IP, i.e. includes infinitely powerful provers.

The idea is to use our computation protocol on each of the circuits specifying the Verifier's computations. More specifically, the following is done:

At each point where the Prover is about to send a message in the original protocol, it will instead send a collection of bit commitments containing the bits of this message. The parties will now use our computation protocol on the circuit specified by the original proof system to be used by the Verifier at this point. This circuit may take as input any message sent so far, but also some secret input - coinflips for example - from the Verifier. The comparability property of the bit commitment schemes will ensure that the same set of messages and input is used consistently throughout. The computation protocol will be executed so that the output of the circuit, i.e. the Verifier's next message in the original proof system, is private to the Prover, by techniques introduced in the previous section. The Prover is now free to read this message and compute a response, perhaps even using infinite computing power.

The last circuit processed is the one doing the computation that the Verifier should use in deciding whether or not to accept the proof in the original proof system. The output of this circuit is opened to the Verifier, who can check that indeed it is "accept".

It is now easy to see that, because the computation protocol is correct, the Verifier can be convinced that it would have accepted, had it done the original protocol with the prover. Note however, that this makes essential use of the fact that one party - in this case the Verifier - can be unconditionally protected. This means that, even using infinite computing power, the Prover will not learn anything about the secrets of the Verifier and is therefore in essentially

the same position as in the original proof system.

On the other hand, since the computation protocol is also secure with respect to the secrets of the Prover, this new proof system is minimum knowledge: the Verifier learns nothing except the fact that  $x \in L$  - everything else is hidden in bit commitments, which are computationally as good as nothing. Note that the number of rounds executed is revealed, and it may therefore be necessary to modify the original proof system such that the number of rounds only depends on the security parameter. This can be done by simply adding “empty” rounds to the original system.

All this can be summarized in the following

**Theorem 8.8.**

Assume that pairs of claw free trapdoor functions exist. Then any language in IP has a minimum knowledge proof system whose running time is polynomially bounded as a function of the running time for the original proof system. Here, the running time is defined to be total number of bit operations executed by the parties during the protocol.

**Proof**

The existence of trapdoor one way functions will make our basic computation protocol work with conditional security, which is what we need for the Prover. The clawfreeness implies the existence of a bit commitment scheme hiding the bits unconditionally [Ch], which means that the Verifier can be protected as required above  $\square$

## 9. Unconditionally Secure Protocols.

In the previous section, we have seen how to implement multiparty computations in the standard model of communication: every message sent becomes known to all participants - although not everybody may be able to decrypt it.

The most general constructions for this model depend on the existence of trapdoor one way permutations, and therefore they must essentially assume that public key cryptography is possible. Moreover, we have also seen that unconditional protection is only possible for *one* participant under this model.

Therefore, two fundamental questions remain:

- 1) What protocol problems can be solved using only private key cryptography, i.e. assuming only authenticated secrecy channels between participants?
- 2) Can unconditional protection be achieved for all participants under a different model?

In recent work by Chaum, Crépeau and the author [ChCrDa], it is shown how to solve in principle any protocol problem, assuming that:

- i) There is an unconditionally secure secrecy channel between every pair of participants.
- ii) If there are  $n$  participants, where  $n = 3d + a$ ,  $a = 1, 2$  or  $3$ , then at most  $d$  participants will deviate from the protocol.

The resulting protocols will be unconditionally secure for all participants, and do not rely on any cryptographic assumptions - the secrecy channels can be implemented using a one time pad, so the really fundamental assumption is not a cryptographic one, but rather one stating that every pair of participants share a sufficient number of random bits initially. Using a crypto-system, one can of course trade this number of bits against the security level of the protocol.

The assumption on the number of cheaters is optimal: in [LaPeSh], it is shown that if more than  $d$  participants cheat, it is not possible to even make the honest participants agree on a message, such as the result of a computation. Further, the constructions have a built-in fault tolerance: the forms of cheating that the protocol can tolerate from at most  $d$  participants include stopping the protocol early.

A detailed description of our results can be found in [ChCrDa], or in Claude Crépeau's phd-thesis [Cr2]. Here, we will just explain the basic ideas very briefly.

A basic ingredient in our construction is Shamir's secret sharing scheme [Sh] which in our situation will be set up as follows: choose a finite field of characteristic 2 with at least  $n+1$  elements. A polynomial  $p(x)$  is said to *determine the secret*  $p(0)$ . We then give to each participant  $P_i$  a *share*,  $p(x_i)$ , where the  $x_i$ 's are distinct non-zero elements. If  $\deg(f) \leq d$ , we obtain that any set of at least  $d+1$  shares determine  $p$  and therefore the secret uniquely, while no information about it is revealed by any set of at most  $d$  shares.

This allows implementation of bit commitments - a participant can commit to a secret value in the field by choosing a random polynomial determining the secret he wants to commit to, and distributing shares of it to every participant. Provided an interactive proof is executed initially to verify the consistency of the shares distributed (see [ChCrDa] for details), the participant is now committed to his secret. Later, a commitment can be opened by having all participants make their shares public.

To implement multiparty computations, we make use of a homomorphic structure of the secret sharing scheme, the usefulness of which was first observed by Benaloh [Be]. Observe that by adding (multiplying) shares of two polynomials  $p$  and  $q$ , we get shares of the polynomial  $p+q$  ( $p \cdot q$ ), which clearly determines the sum (product) of the original secrets.

The basic idea is now very simple: each participant will distribute shares of his private input, and we then do the computation by having each participant do a corresponding computation on his shares, and make the final result public. There are two problems with this idea:

- 1) We cannot trust every participant to do the computation correctly. Therefore participants must commit, not only to their own inputs, but to all the shares they receive, so that they can prove that they follow the protocol.
- 2) Multiplication of two polynomials of degree at most  $d$  yields a polynomial of degree at most  $2d$ , so it takes only very few multiplications before the number of shares available is not sufficient to determine the polynomial uniquely. We therefore introduce a procedure which after the multiplication replaces the product with a randomly chosen polynomial of degree at most  $d$  determining the same secret as did the product.

Details on the solution to the two problems can be found in [ChCrDa]. Very recently, Crépeau found a way to combine the protocol from the previous section with the protocol mentioned here. Recall that in the model from the previous section, one could get only conditional security, with fault tolerance against up to  $n/2$  cheaters. With the present protocol, one can tolerate essentially  $n/3$  cheaters, but with unconditional security. The combined protocol has both forms of security: if there are  $C$  cheaters with  $C < n/3$ , then we have unconditional security, while if  $n/3 < C < n/2$ , the security is conditional. The basic idea here is to let the secrets that are unconditionally hidden be the messages sent in the protocol from Section 8. Thus if more than  $d$  participants collaborate, all they get will be encryptions of other peoples' secrets.



### Table of Abbreviations.

DLF	The Discrete Log Family, page 9.
DLS	The Discrete Log Bit-Commitment Scheme, page 31.
IAF	The Intractability Assumption on Factoring, page 9.
IAF2	The Intractability Assumption on Factoring, page 14.
IDL	The Intractability Assumption on Discrete Log, page 10.
JSF	The Jacobi Symbol Family, page 15.
JSS	The Jacobi Symbol Bit-Commitment Scheme, page 30.
SQF	The Squaring Family, page 9.
SQF2	The Squaring Family II, page 12.
QRA	The Quadratic Residuosity Assumption, page 11.
QRF	The Quadratic Residue Family, page 10.
QRS	The Quadratic Residuosity Bit-Commitment Scheme, page 29.

## References.

- [AdHu] L.Adelman and M.Huang: "Recognizing Primes in Random Polynomial Time", Proc. of STOC 87.
- [Ba] E. Bach: *Analytic Methods in the Analysis and Design of Number-Theoretic Algorithms*, ACM Distinguished Dissertation, 1984, MIT Press.
- [Be] J.Benaloh: "Secret Sharing Homomorphisms", Proc. of Crypto 86, Springer.
- [BePe] Berstel and Perrin: *The Theory of Codes*, Academic Press, 1985.
- [Bo] B.den Boer: Private communication.
- [BCDG] E.Brickell, D.Chaum, I.Damgård and J. van de Graaf: "Gradual and Verifiable Release of a Secret", Proc. of Crypto 87, Springer.
- [Bl] M.Blum: "Three Applications of the Oblivious Transfer", Dept. of EECS, Univ. of California, Berkeley, 1981.
- [Bl2] M.Blum: "How to Exchange (secret) Keys", *ACM Transactions on Computer Systems*, vol. 1, 1983, pp.175-193.
- [BlMi] M.Blum and S.Micali: "How to Generate Cryptographically Strong Sequences of Pseudorandom Bits", *SIAM Journal on Computing*, vol. 13, 1984, pp.850-864.
- [BrChCr] G.Brassard, D.Chaum and C.Crépeau: "Minimum Disclosure Proofs of Knowledge", technical report PM-8710, 1987, CWI, Amsterdam.
- [BrCr] G.Brassard and C.Crépeau: "Non-transitive Transfer of Confidence: a *perfect* zero-knowledge Protocol for SAT and beyond", Proc. of FOCS 86, pp.188-195.
- [BrCrRo] G.Brassard, C. Crépeau and J. Robert: "Reductions among Disclosure Problems", Proc. of FOCS 86.
- [BoKrKu] J.Boyar, M.Krentel and S.Kurtz: "A Discrete Logarithm Implementation of Zero-Knowledge Blobs", technical report, dept. of Computer Science, University of Chicago, 1987.
- [Ch] D.Chaum: "Demonstrating that a Public Predicate can be Satisfied without Revealing any Information about how", Proc. of Crypto 86, Springer.
- [Ch2] D.Chaum: "How to Keep a Secret Alive", Proc. of Crypto 84, Springer.
- [CEGP] D.Chaum, J.Evertse, J. van de Graaf and R.Peralta: "How to Demonstrate Possession of a Discrete Log Without Revealing it", Proc. of Crypto 86, Springer.
- [ChDaCr] C.Chaum, I.Damgård and C.Crépeau: "Multiparty Unconditionally Secure Protocols", presented at rump session of Crypto 87, to appear in Proc. of STOC 88.
- [ChDaGr] D.Chaum, I.Damgård and J.van de Graaf: "Multiparty Computations Ensuring Privacy of Each Party's Input and Correctness of the Result", Proc. of Crypto 87, Springer.

- [ChEv] D.Chaum and J.Evertse: "A Secure and Privacy Protecting Protocol for Transmitting Personal Information Between Organisations", Proc. of Crypto 86, Springer.
- [ChGr] D.Chaum and J.van de Graaf: "An Improved Protocol for Demonstrating Possession of a Discrete Log", Proceedings of Eurocrypt 87, Springer.
- [Cr] C.Crépeau: "Equivalence Between Two Flavours of Oblivious Transfer", Proc. of Crypto 87, Springer.
- [Cr2] C.Crépeau: phd- Thesis, MIT, 1986.
- [Co] D.Coppersmith: "Another Birthday Attack", Proc. of Crypto 85, Springer.
- [CoFi] J.Cohen and M.Fisher: "A Robust and Verifiable Cryptographically Secure Election Scheme", Proc. of FOCS 85, pp.372-383.
- [Da] I.Damgård: "Collision Free Hash Functions and Public Key Signature Schemes", Proc. of Eurocrypt 87, Springer.
- [De] D.Denning: "Digital Signatures with RSA and Other Public Key Crypto-Systems", *CACM*, vol.27, 1984, pp.441-448.
- [Di] J.Dixon: "The Probability of Generating The Symmetric Group", *Math. Z.*, vol. 110, 1969, pp.199-205.
- [DP] D.Davis and W.Price: "The Application of Digital Signatures Based on Public Key Crypto-Systems", Proc. of CompCon 1980, pp.525-530.
- [GaHaYu] Z.Galil, S.Haber and M.Yung: "Secure and Fault Tolerant Protocols in the Public Key Model", Proc. of Crypto 87, Springer.
- [GGM] O.Goldreich, S.Goldwasser and S.Micali: "How to Construct Random Functions", Proc. of FOCS 84, pp.464-479.
- [Gor] J.Gordon: "Strong Primes are Easy to Find", Proc. of Crypto 84, Springer.
- [Go] O.Goldreich: "Two Remarks Concerning the GMR Signature Scheme", Proc. of Crypto 86, Springer.
- [GoMi] S.Goldwasser and S.Micali: "Probabilistic Encryption", *JCSS*, vol.28, No.2, 1984, pp.270-299.
- [GoSi] S.Goldwasser and M.Sipser: "Arthur Merlin Games versus Interactive Proof Systems", Proc. of STOC 1986, pp.59-68.
- [GoVa] O.Goldreich and V.Vainish: "How to Solve any Protocol Problem; an Efficiency Improvement", Proc. of Crypto 87, Springer.
- [GoMiRa] S.Goldwasser, S.Micali and C.Rackoff: "The Knowledge Complexity of Interactive Proof Systems", Proc. of STOC 85, pp.291-304.
- [GoMiRi] S.Goldwasser, S.Micali and R.Rivest: "A "paradoxical" Solution to the Signature Problem", Proc. of FOCS 1984, pp.441-448.

- [GoMiRi2] S.Goldwasser, S.Micali and R.Rivest: "A Signature Scheme Secure Against Adaptive Chosen Message Attacks", to appear.
- [GoMiWi] O.Goldreich, S.Micali and A.Wigderson: "How to Play any Mental Game", Proc. of FOCS 87.
- [GoMiWi2] O.Goldreich, S.Micali and A.Wigderson: "Proofs that Yield Nothing but the Validity of the Assertion, and the Methodology of Cryptographic Protocol Design", Proc. of FOCS 86.
- [GrPe] J.van de Graaf and R.Peralta: "A Simple and Secure Way to Show the Validity of Your Public Key", Proc. of Crypto 87, Springer.
- [HaSh] J.Hastad and A.Shamir: "The Cryptographic Security of Truncated Linearly Related Variables", Proc. of STOC 85, pp.356-362.
- [Kr] E.Kranakis: *Primality and Cryptography*, Wiley-Teubner Series in Computer Science, 1986.
- [LaPeSh] L.Lamport, Shostak and Pease: "The Byzantine Generals Problem", *ACM Trans. on Programming Languages and Systems*, vol.4, No.3, 1982, pp.382-401.
- [LuMiRa] M.Luby, S.Micali and C.Rackoff: "How to Simultaneously Exchange a Secret Bit by Flipping a Symmetrically-Biased Coin", Proc. of FOCS 83, pp.11-21.
- [Ra] M.Rabin: "How to Exchange Secrets by Oblivious Transfer", Technical Memo. TR-81, Aiken Computation Laboratory, Harvard University, 1981.
- [Sh] A.Shamir: "How to Share a Secret", *CACM*, vol.22, no.11, 1979, pp.612-613.
- [St] J.Stephens: "Lentra's Factoring Method Based on Elliptic Curves", Proc. of Crypto 85, Springer.
- [Te] T.Tedrick: "How to Exchange Half a Bit", Proc. of Crypto 83, Plenum Press, pp.147-151.
- [Te2] T.Tedrick: "Fair Exchange of Secrets", Proc. of Crypto 84, Springer, pp.434-438.
- [VaVa] V.Vazirani and V.Vazirani: "Trapdoor Pseudo Random Number Generators with Applications to Cryptographic Protocol Design", Proc. of FOCS 83, pp.23-30.
- [We] Wegman and Carter: "New Hash Functions", *JCSS*, vol. 22, 1981.
- [Wi] Winternitz: "Producing a One Way Hash Function from DES", Proc. of Crypto 83, Springer.
- [Ya2] A.Yao: "Protocols for Secure Computations", Proc. of FOCS 82.
- [Ya] A.Yao: "How to Generate and Exchange Secrets", Proc. of FOCS 86.
- [YuIm] M.Yung and R.Impagliazzo: "Direct Minimum Knowledge Computations", Proc. of Crypto 87, Springer.