Direct Minimum-Knowledge Computations

(Extended Abstract)

Russell Impagliazzo
U.C. Berkeley

Moti Yung[1]
Columbia University

Abstract

We present a protocol scheme which *directly* simulates any given computation, defined on any computational device, in a *minimum-knowledge* fashion. We also present a scheme for simulation of computation in *dual* (perfect) *minimum-knowledge* fashion. Using the simulation protocol, we can assure that one user transfers to another user exactly the result of a given computation and nothing more.

The simulation is direct and efficient; it extends, simplifies and unifies important recent results which have useful applications in cryptographic protocol design. Our technique can be used to implement several different sorts of transfer of knowledge, including: transfer of computational results, proving possession of information, proving knowledge of knowledge, gradual and adaptive revealing of information, and commitment to input values.

The novelty of the simulation technique is the separation of the data encryption from the encryption of the device's structural (or control) information.

[1] Supported in part by NSF grants MCS-8303139 and DCR-8511713 and an IBM graduate fellowship.

C. Pomerance (Ed.): Advances in Cryptology - CRYPTO '87, LNCS 293, pp. 40-51, 1988.

1. Introduction

Zero-knowledge interactive proof-systems are a new technique which can be used as a cryptographic tool for designing provably secure protocols. Goldwasser, Micali, and Rackoff originally suggested this technique for controlling the knowledge released in an interactive proof of membership in a language, and for classification of languages [19]. In this approach, knowledge is defined in terms of complexity theory, and a message is said to convey knowledge if it gives a computational advantage to the receiver, for example by giving him the result of an intractable computation. The formal model of interacting machines is described in [19, 15, 17].

A *proof-system* (for a language L) is an interactive protocol by which one user, the *prover*, attempts to convince another user, the *verifier*, that a given input x is in L. We assume that the verifier is a probabilistic machine which is limited to expected polynomial-time computation, while the prover is an unlimited probabilistic machine. (In cryptographic applications the prover has some trapdoor information, or knows the cleartext of a publicly known ciphertext.)

A *correct* proof-system must have the following properties:

- If $x \in L$, the prover will convince the verifier to accept the proof with very high probability.
- If $x \notin L$ no prover, no matter what program it follows, is able to convince the verifier to accept the proof, except with vanishingly small probability.

The above definition can be extended to correctness of a more general protocol which transfers to the verifier the result of a computation. The possible results transferred by the protocol form a probability distribution defined by the computation of the probabilistic machines. Such a protocol is *correct* if the prover can transfer a correct result according to the specified probability distribution (with very high probability), and no adversary can transfer an incorrect result by biasing the distribution (except with a vanishingly small probability).

In order to model the fact that the interacting parties are not memory-less and, for example, may choose their action according to previous events, they are provided with *history* tapes. Each user's history includes the messages exchanged, the random bits used and the private output that he computes during the execution.

A proof-system is *zero knowledge* if the following holds. Given any input $x \in L$, and an initial verifier's history h (representing the context in which the protocol is started), any verifier can generate a probability distribution of possible transcripts of a simulated history of an interaction. The simulation should not be distinguishable by any polynomial time computation from the history of actual interactions (of the prover and this verifier) on the same input x and same initial history h. The definition of *polynomial time indistinguishability* is given in [23, 19]. The zero-knowledge property implies that the interaction gives the verifier the intended result and nothing more, a fact which is crucial in controlling the knowledge transmitted in cryptographic protocols.

The zero-knowledge notion has been useful in designing secure protocols; see for example [19, 13, 11, 2]. In [15] the notion was extended to deal with the knowledge released in a more general protocol which transfers a computational result; an example was given of a protocol for transferring the value of a certain number-theoretic predicate.

For a protocol which transfers a result, we say that the prover gives the computational result to a verifier in a *minimum-knowledge* fashion if the verifier, when he is allowed to get one answer from a *result oracle*, can generate a simulated transcript of the history of an interaction which is indistinguishable from the history of an actual interaction with the prover which gives him this result. This was originally formalized for result transfer protocols in [15]: if the provision of the oracle's answer enables the verifier to simulate the entire interaction, then we can say that the interaction itself did not give any additional knowledge to the verifier.

An important result by Goldreich, Micali and Wigderson [17] shows that, under the assumption that a one-way permutation exists, all NP languages have zero-knowledge proofs. This result is the starting point of our work. As described in [17], this result has important consequences for the design of provably secure protocols, as will be explained in the next section.

The result of [17] is proved by exhibiting a protocol for graph 3-colorability, i.e. a proof-system for the language of 3-colorable graphs. To prove a general NP statement one first has to translate it into an instance of the graph 3-colorability problem (using the publicly known Karp reduction [16]). Recently, several protocols for other NP languages were suggested, using similar techniques which exploit properties of specific NP-complete languages. In [8], under the assumption that quadratic residuosity is intractable, a protocol for satisfiability (SAT) was suggested which directly simulates the circuit that evaluates given instances of SAT. Chaum [10], independently, gave a protocol for direct circuit simulation under the assumption that "claw-free" functions exist. In [17] it is mentioned that the technique of direct computation in [8] does not seem to generalize to an arbitrary one-way function. Furthermore, it seems that the techniques of [10, 8] do not generalize to a direct simulation of a general computational device, since they rely on the input-output relationships of circuit gates.

In this paper we show how it is possible to directly simulate the computation of any given computational device. The scheme presented is correct; if a one-way permutation exists, then the scheme guarantees that the transfer is done in minimum-knowledge fashion. Our construction applies to any polynomial-size device (such as a polynomial-size circuit, or a polynomial-time non-deterministic or probabilistic-non-deterministic Turing machine [20]). The minimum-knowledge property assures that any information about the input which is not revealed by the output remains secure. Using this new technique, one can avoid the Karp reduction to a specific NP-complete problem, and directly prove the result of the computation.

The efficiency of our protocol scheme is directly related to the computational complexity of the given problem. When the computation is given by a computing circuit of size c (or a Turing machine in which the product of time and space is c) and k is the size of an encryption of one bit (which is equal to the system's security parameter), the message size required is ck. In order to make the protocol correct with high probability, we have to pay in communication rounds. If the tolerated error probability of the protocol is $\delta(k)$, then r, the number of rounds in our scheme, must be $\log\delta(k)$. In case one wants an exponentially small probability, then $r=k$; if δ has to vanish faster than any k^{-n} for all n, then $r=\log^{1+\epsilon}k$ is enough. Thus, the total communication complexity is ckr, where k is the overhead for encryption and r is the overhead for confidence.

Our protocol scheme applies to several different notions of "minimum-knowledge computations" suggested recently. It is a uniform method which can be used in the following interactions:

- It enables the prover to simulate a given computation and transfer only its result [19, 15, 17]. The prover can transfer only parts of the computation (parts of the input, output or intermediate results), hiding all other information from the verifier. It seems that exhibiting such partial relations in a minimum-knowledge fashion is harder to do using the reduction of the computation to an NP statement, while in direct computation, partial data decoding is easily achievable, and is minimum-knowledge with respect to a 'result oracle' which gives the verifier these partial data.
- It can be used to convince the verifier that the prover possesses a piece of information. This is similar to what was formalized as proving "possession of information" [22], and proving "knowledge of knowledge" [12].
- It has a significant advantage over NP-reduction in solving the problem introduced here of revealing the output information in an "interactive gradual fashion"; that is, when bits of the results are being revealed gradually one by one by the prover, for example, in exchange for an appropriate payment by the verifier. The protocol is run once and for all, and later only the result bits are opened gradually; no extra interaction is needed. In the full paper we show how to do this and how to perform "adaptive gradual trading of a result", in which a verifier may decide which partial result he wants based on previously opened bits. This adaptive gradual opening of information is suggested and solved here. The difficulty is that, in order to make sure that such an interaction is minimum-knowledge, the simulating verifier should receive only the required information which is actually opened, and in an adaptive fashion.
- It can be used directly to verify a "commitment to a value". This is a model of computation with encrypted data presented by Yao [24]. In this model, a prover commits himself to certain data by announcing an encrypted version of it. Later, he can convince the verifier that a computational result transferred indeed used as inputs the cleartext values to which he had committed himself.
- It can also simulate the most general interactive proof (IP) in a minimum-knowledge fashion. The technique is applied to the probabilistic nondeterministic polynomially bounded Turing machine [20] which models the general interactive proof. The fact that the general interactive proof can be done in minimum-knowledge was first observed by Ben-Or.

The *dual* (or perfect) *zero-knowledge* notion was suggested by Brassard and Crepeau, and by Chaum [7, 10]. The prover is restricted to polynomial time; however he knows a witness to the given NP statement, or an input to the given computation. The verifier may have unlimited computing power (or, in a variation on this model, we assume that he is limited to polynomial-time), and he wants to be convinced that the prover has a witness.

The differences between the dual model and the model described above (which we may call the *primal* model) are:

- The computational power of the parties is interchanged.
- The verifier accepts the proof only under a number-theoretic or cryptographic assumption in the dual model and unconditionally in the primal one.
- The computation simulating the actual interaction in the dual model produces a transcript of interaction which is identical to the original one (and thus indistinguishable); such a protocol is called *perfectly zero-knowledge*. In the primal model the transcript of the interaction is not

identical, and is indistinguishable only under the computational complexity assumption.

- The effect of deviations from the specified computational power of the parties differs in the two models. In the primal model, if the verifier is given enough computational resources before the interaction (say, exponential time), the interaction is still (trivially) minimum-knowledge (using a simulator with the same resources as the verifier). In the dual model, when the prover gets enough time before the interaction, he can cheat and the protocol might not be correct. After the interaction, on the other hand, in the primal model giving enough time to the verifier may enable him to extract more knowledge than intended; in the dual model the verifier cannot extract any extra knowledge when given unlimited time after the interaction.

Assuming there exists a one-way function which is a group homomorphism, we give a direct simulation for the dual minimum-knowledge model. Notice that all known number-theoretic permutations which are assumed to be one-way (based on the problems of RSA, quadratic residuosity, and discrete logarithm) are group homomorphisms. The proof-system protocol for this model is similar to the first simulation protocol; thus we present a unified way to treat the two models. We note that the dual zero-knowledge model protocols in [7, 10] are based on quadratic residues or claw-free pairs of functions, and they require the ability to prove equality of two encrypted bit values in a minimum-knowledge fashion. We do not need this property for our protocol. The only requirement is that the homomorphism is one-way; it is used in an encoding technique we call *locking*. The simulation technique in the dual model applies to all the various notions of "minimum-knowledge computations" presented above which are applicable to this model.

In this abstract we present only the basic protocols for simulation of circuit computations. The precise definitions of the various notions (such as "minimum-knowledge" and "transfer of knowledge") will be given in the full paper.

2. Applications of Zero-Knowledge Computations

We assume that a probabilistic encryption scheme exists. Yao [23] showed that this is implied by the existence of a one-way permutation. Examples of concrete encryption schemes based on assumptions of the intractability of certain number-theoretic problems are given in [18, 3, 4, 1]. In the full paper we formalize exactly what properties are required of the encryption scheme in order to implement our constructions.

When a cryptographic protocol is performed, each user wants to be sure that his partners follow the protocol. For example, in a "mental-poker" game a player wants to know that his opponent follows his instructions as specified by the protocol, and gets legal cards from the deck. The fact that every NP language has a zero-knowledge proof makes on-line validation of such facts possible [17]. Given a probabilistic encryption scheme, the set of valid encryptions of a legal message is an NP language. Thus, users can check interactively in minimum-knowledge fashion that a message sent during an execution of a protocol was indeed generated as specified by the protocol. We call protocols in which each message is checked on-line using zero-knowledge proofs *validated* protocols. This validation can be the bottleneck in the protocol. Therefore, to make validated protocols efficient, the proof-systems used throughout the protocol should be as efficient as possible. Our direct computation technique can speed up the validation proofs.

Here we present a few examples of validations. Assume that a user is supposed to choose a number which is a product of m primes; he constructs a probabilistic circuit which checks primality of m inputs, then multiplies them and gives the product as an output. Afterwards, he can perform a minimum-knowledge simulation of the computation and open only the output.

In another example, suppose that a user sends a value $y=f(x)$ and wants to show that he knows the argument x of the one-way function f, while keeping x secure. He can convince another user by a minimum-knowledge simulation of the circuit which computes $f(\cdot)$. The simulation itself uses a one-way function to encode the circuit, which we may call the underlying encryption function. (In fact, we can use a 'bootstrapping technique' in which the circuit computation that represents the computation of f is performed using f itself as the underlying function). In the next section we explain how these computations are done.

In the above example, the result of the computation is the only information opened by the user. It may be the case that this information has to be opened gradually, and the receiver of the information has to pay for each result bit (for example by revealing bits of his own secret result). This is the ''gradual interactive revealing of information'' that we introduce in this work (to be explained in the full paper). The notion is similar to the one presented in [9], but we limit the interaction to an initial phase, separate from the actual opening of the bits. We also present an adaptive version where the bits to be revealed are decided on-line by the verifier.

Assume that a relation $Q=(\cdot,\cdot)$ and a public input x are given. A prover wants to demonstrate that he ''knows'' or possesses [22] a private witness w such that $(x,w)\in Q$. For example, the witness can be a certificate of the membership of x in an NP language. The possession of the witness can be proved using a minimum-knowledge simulation of the computation of the predicate Q. Similarly, the prover can demonstrate that he has a witness w either to the fact that $x\in L$ or to the fact that $x\notin L$, where $L\in$ NP $\cap$ co-NP, without revealing to the verifier which is the case. There is a a predicate $A(x,w)$ for L, and a similar predicate $B(x,w)$ for the complement language $\overline{L}$. The prover has to convince the verifier that he has a witness which satisfies the predicate $A(x,w)\vee B(x,w)$; he does this by executing a minimum-knowledge simulation of the computation of this predicate.

Suppose a prover wants to demonstrate that he computes $C(x)$ with a value x that he knows, and to which he has committed himself [24]. The commitment is done at the beginning of the protocol, when the prover sends y, where $y=f(x)$ is the image of x under a one-way function. Then, using the same input x, he evaluates both $f(\cdot)$ and $C(\cdot)$. The prover opens the outputs, which have to be y and the result $C(x)$. The verifier is convinced that the computation must have used the input to which the prover committed himself earlier.

3. Direct Minimum-Knowledge Computations

In this section, we show how any one-way permutation may be used in *direct* minimum-knowledge interactive simulation of a computation. Let P be the prover and V the verifier.

The essential idea of the protocol is that the prover constructs and sends to the verifier a copy of a simulation of the computing device (i.e. a copy of a circuit or of a computation history of a Turing machine). This copy includes encoding of the possible input, intermediate results, and output data. In addition it includes encoding of structural information about the computing device. In the case of a circuit, the structural information consists of connections (pointers) from the output of one gate to the input of another gate in the circuit (based on the gates' truth tables); a pointer connects two entries which encode the same bit value. For a Turing machine, the structural information consists of connections between two consecutive instantaneous descriptions in the computation history; these connections are based on the machine's finite control (transition table). Any computation is a combination of some local data manipulation and transitions, based on the device's finite control, which direct the computation to its next step. Our technique separates the encoding of data from the encoding of the possible continuations given by the control element. We describe here only the circuit model computations.

First we outline the protocol. Upon receiving an encoding of the circuit, the verifier flips a coin and chooses one of two options. With probability 1/2 he decides to *verify*, that is to request that the prover open all the encryptions in the copy of the circuit. In this case he can check that the construction is a legal computing circuit with the right data in each gate, and that the structural information encodes correct connections between gates. With probability 1/2 the verifier chooses to *compute*, in which case the prover opens only the result of the computation. To prove that the output presented is in fact the computed result of the circuit, the prover opens only the cleartexts of the pointers connecting entries which are involved in the computation, while all other information is left encrypted. The connections show how to navigate through the circuit from the input data to the output gates. Then the circuit's output is opened as well. The unopened information appears random (and is polynomial-time indistinguishable from a set of encryptions of any fixed arbitrary data).

The process is repeated r times; each time the prover is ready either to verify or to compute. If all verifications are successful (that is, each circuit encoding does simulate the computing circuit), and all the computations produce a connection to the same opened output, then the verifier accepts the result of the computation.

In a computation, only pointers between gates leading to the output are opened. We remark that if there are random inputs in the circuit, then from the two possible elements in an input entry V chooses at random which element to use, and P uses these elements in the computation. (V is actually flipping a coin into P's well [5].) In the direct simulation, the result of the computation need not be restricted to an output of the given circuit. It can include relations between parts of the inputs, intermediate results, and outputs.

Below we briefly describe the circuit design. (We will give a detailed description in the next version.)

- A circuit encoding consists of gates corresponding to the actual gates of the given circuit.
- Each gate has input entries, a truth table and output pointers.
- Each input entry represents an input bit to the gate and consists of an unordered pair of ciphertexts encrypting 0,1 or 1,0 at random. In an actual computation one of the two ciphertexts is chosen.
- The truth table represents the computation done by the gate. It consists of rows; each one maps a combination of entries' ciphertext values to an output pointer. (For example, a binary

gate has four rows.)

- The rows of each truth table are permuted at random.
- The pointer in each row is a reference to the input entry of the next gate in the circuit. The pointer value is either *first* (an encryption of 0) or *second* (an encryption of 1).
- If the pointer value is *first* (0) it means that the output value of this row is the same as the value of the first ciphertext in the input entry of the next gate, while a pointer of value *second* (1) means that the output is the same as the second value in the input entry of the next gate.
- Since the entries and rows in the table are randomly permuted in each gate, the pointer connection giving the structural information about the circuit appears random when it is deciphered while keeping the other information encrypted.
- If the gate does not connect to another gate but actually gives a circuit output bit, then the pointer's value (i.e. the bit it encrypts) is the actual output value.

Next we briefly describe the protocol.

PROTOCOL 1:

{The prover P (who uses his probabilistic encryption procedure E) and the verifier V simulate the computation of a circuit C in a minimum-knowledge fashion.}

repeat *r* times {loop}:

1. P probabilistically encrypts C as described above. Let E(C) denote the encryption. P → V: "E(C)".

2. V chooses a bit $b \in \{0,1\}$. V → P: "b".

3. If b=0 then
{*verify*}: P opens all the encryptions of all gates in E(C) and sends the cleartext circuit to V.
else
{*compute*}: P opens only the pointers of the specific computation and sends the cleartexts of these pointers to V (including the outputs).

4. If b=0 V verifies that C is properly encrypted;
if b=1 he verifies that the opened pointers lead from the (unopened) input entries to the output pointers.

{end loop}

If all computations give the same value and all verifications are successful then V accepts. In any other case he rejects.

{END PROTOCOL 1}

Theorem 1: Assume that a one-way permutation exists. There is a direct minimum-knowledge simulation of any circuit of polynomial size *c*. It takes *r* rounds and uses *kc*-bit messages, where *k* is the security parameter and *c* is the size of the simulated circuit. The error probability is $1-(1/2^r)$.

The protocol is a correct result-transfer protocol because:

- The prover is able to transfer the result and compute the encrypted circuit transferring the result.
- On the other hand, no prover can cheat the verifier since in order to do this without being caught he has to guess the r random coins of step 2 and this event has vanishingly small probability ($1/2^r$).

Notice that with very high probability ($1 - 1/2^r$) any machine P' that successfully performs the prover's role is ready, in at least one of the iterations, to verify that indeed he constructed the circuit simulation properly, but is asked instead to compute. This means that in at least one of the rounds the prover indeed had a valid circuit and indeed computed using a witness. Since all computations and verifications were valid, the verifier accepts the correctness of the computation. This is true also when the computational result is actually not interesting, but the circuit is used only for a demonstration of some computational power as was formalized in [12, 22] (e.g. "knowledge of a witness" as in one of the examples of section 2).

The system is proven to be minimum-knowledge by showing that for any given verifier V' there is a simulating polynomial-time machine. The machine gets the result of the computation from a 'result oracle' (as was described in [15]) and produces (in expected polynomial time) an output which is a simulated history of the interaction; this output is polynomial-time indistinguishable from a history of V' recorded in an actual interaction between V' and the prover.

The crux of the proof is that the unopened part of an actual interaction is indistinguishable from an encryption of fixed random data in the simulated transcript. No polynomial time procedure can get any partial information about this unopened ciphertext of the transcript in order to tell whether it is an actual or a simulated one. If there is a polynomial time procedure which does, then the set of transcripts is a message space for which it is possible to distinguish between encrypted messages in polynomial time. These messages are probabilistically encrypted based on a one-way function. Such a procedure could be converted into a procedure which efficiently inverts the one-way function, but this is assumed to be impossible [18, 23, 21].

4. Direct Dual (Perfect) Minimum-Knowledge Computations

The direct minimum-knowledge computation technique presented is also applicable to the dual model of minimum-knowledge protocols. In this model the proof is presented by a polynomial-time prover to a powerful verifier, and the proof is accepted only under a cryptographic assumption [7, 10]. The proof has to convince the verifier that the prover possess some computational knowledge. Chaum [10] uses this model for identification protocols in the context of his credential mechanism.

Assume there is a one-way function f which is a group homomorphism. (All the number-theoretic one-way functions discussed in the literature are group homomorphisms: RSA, modular exponentiation, modular squaring). Then the following holds:

- Given a random ciphertext value $f(x)$, no polynomial time procedure can find x. This follows from the assumption that the function is one-way.
- Given a random ciphertext value $f(x)$, and a random value q in the range of f. It is impossible

(even with unlimited computing power) to decide whether q was generated as $q=f(x)*f(s)$ for some randomly chosen s, or $q=f(t)$ for some randomly chosen t.

- A random polynomial time procedure which, on input $f(x)$, is able to compute q, s, and t related as above, is able to compute x as well. This would contradict the assumption that f is one-way.

These facts are the basis of the following protocol scheme. Previously, the dual minimum-knowledge simulated computation protocols [10, 7, 6] needed the ability to provide a *zero-knowledge proof of equality* of two cleartexts, given the ciphertexts. Here we show that, using the new technique, this requirement is not needed.

PROTOCOL 2:

{The polynomial time prover P and the verifier V simulate the computation of a circuit C.}

1.V → P: " $z=f(x)$ ".

{It is possible to have a model of computation in which z is given and x is unknown to both parties. On the other hand, it might be the case that the verifier has unlimited power and knows x, in which case the above transmission can be followed by a minimum-knowledge proof in which V validates that the value z is in the range of f and x is kept secret. This is done using the zero-knowledge proof of a preimage of a group homomorphism of [14] (generalizing the proof of quadratic residuosity of [19]).}

P encrypts C as described in section 3 above, using as encryption function the following:
0 is encoded as $f(t)$, for a randomly chosen t,
1 is encrypted as $z*f(s)$, for a randomly chosen s.

{Notice that in the above encryption methods, the encodings of 1 and 0 are drawn from the same probability distributions. The powerful machine V cannot distinguish between an encryption of 0 and an encryption of 1. The only difference between them is the way they are produced by the prover. We call this encryption method *locking*, since it is the computation by P which commits the ciphertext to a value, and he is the only one who can unlock and reveal the original value.}

{Exhibiting a cleartext value for a ciphertext y is done by opening (*unlocking*), which means giving either a preimage by f of y (to unlock y as an encryption of 0) or a preimage of $y*z^{-1}$ (to unlock y as an encryption of 1).}

2. Using the locking of bits as encryption mechanism, the users follow the loop and the acceptance procedure of protocol 1.

{END PROTOCOL 2}

Theorem 2: Assume that a one-way group homomorphism exists. There is a direct 'dual zero-knowledge' simulation of any circuit of size c. It takes r rounds and uses kc-bit messages, where k is the security parameter. Its error probability is $1-(1/2^r)$.

The proof relies on the properties of the locking method described above. The fact that locked 0 and locked 1 are identical (absolutely indistinguishable) makes the set of transcripts output by the simulating machine *identical* to the set of histories of a real interaction between a verifier and the prover. Such a protocol is called "perfectly minimum-knowledge".

The verifier accepts the proof under the assumption that the prover cannot invert $f(x)$, and when unlocking a value, say 1, he can open it only as 1. Notice that as in the proof of theorem 1, with very high probability, in one of the iterations a prover P' is ready to verify the circuit construction, but is asked, instead, to compute; thus V accepts with very high probability. A prover cannot cheat since he is likely to be caught with very high probability. This shows that the protocol is correct.

5. Conclusions

Minimum-knowledge interaction seems to be an important tool for the implementation of secure correct protocols. We presented direct minimum-knowledge interactive computation systems. Our implementation is easily derived from the specification of the computational problem; it is simple and efficient. It is appropriate for all of the models of minimum-knowledge interaction that have recently been proposed, as well as the new ones proposed here.

The technique of direct computation deals in a uniform way with any computational problem, and applies uniformly to the two models of zero-knowledge interactions and to the various notions of interactive knowledge-transfer.

Acknowledgments

We wish to thank Manuel Blum, Gilles Brassard, David Chaum, Oded Goldreich, Stuart Haber, Steven Rudich, and Mike Sipser for their helpful discussions and comments.

References

1. Alexi, W., Chor, B., Goldreich O. and Schnorr C.P. RSA/Rabin Bits are 1/2 + (1/poly(k)) Secure. Proc. 25th FOCS, IEEE, 1984, pp. 449-457.

2. Benaloh, J.C. and Yung M. Distributing the Power of a Government to Enhance the Privacy of Voters. Proc. 5th PODC, ACM, 1986, pp. 52-62.

3. Blum, M. and S. Goldwasser. An Efficient Probabilistic Public-Key Scheme Which Hides All Partial Information. Proceedings of Crypto84, 1985, pp. 289-301.

4. Blum, L., Blum M. and Shub M. Comparison of Two Pseudo-Random Number Generators. Proceedings of Crypto82, August, 1982, pp. 61-78.

5. Blum, M. Coin Flipping by Phone. COMPCON, IEEE, 1982, pp. 133-137.

6. Boyar, J.F., M.W. Krentel, and S.A. Kurtz. A Discrete Logarithm Implementation of Zero-Knowledge Blobs. 87-002, University of Chicago, March, 1987.

7. Brassard, G. and C. Crepeau. Non-Transitive Transfer of Confidence: A Perfect Zero-Knowledge Interactive Protocol for SAT and Beyond. 27th FOCS, IEEE, October, 1986, pp. 188-195.

8. Brassard, G., and Crepeau C. Zero-Knowledge Simulation of Boolean Circuits. Proceedings of Crypto 86, 1986.

9. Brickell, E.F., D. Chaum, I. Damgard, and J. van de Graaf. Gradual and Verifiable Release of a Secret. These proceedings.

10. Chaum, D. Demonstrating that a Public Predicate can be Satisfied Without Revealing Any Information About How. Proceedings of Crypto86, 1986.

11. Cohen, J.C. (Benaloh) and Fischer M.J. A Robust and Verifiable Cryptographically Secure Election Scheme. Proc. 26th FOCS, IEEE, 1985, pp. 372-383.

12. Feige, U., A. Fiat and A. Shamir. Zero-Knowledge Proofs of Identity. 19th STOC, 1986, pp. 210-217.

13. Fischer, M., S. Micali, C. Rackoff, and D. Wittenberg. An Oblivious Transfer Protocol Equivalent to Factoring. Manuscript, 1986.

14. Galil, Z. , Haber S. and Yung M. Symmetric Public-Key Encryption. Crypto85 proceedings, 1985, pp. 128-137.

15. Galil, Z., Haber S. and Yung M. A Private Interactive Test of a Boolean Predicate and Minimum-Knowledge Public-Key Cryptosystems. Proc. 26th FOCS, IEEE, 1985, pp. 360-371.

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

17. Goldreich, O., S. Micali and A. Wigderson. Proofs that Yield Nothing But their Validity and a Methodology of Cryptogrphic Protocol Design. 27th FOCS, IEEE, October, 1986, pp. 174-187.

18. Goldwasser, S. and Micali S. Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information. Proceedings of the 14th Annual ACM Symp. on Theory of Computing, ACM-SIGACT, May, 1982, pp. 365-377.

19. Goldwasser, S., S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proof-Systems. 17 STOC, ACM-SIGACT, May, 1985, pp. 291-304.

20. Goldwasser, S. and M. Sipser. Private Coins versus Public Coins in Interactive Proof Systems. Proceedings of the 18 Annual ACM Symp. on Theory of Computing, ACM-SIGACT, May, 1986, pp. 59-68.

21. Micali, S., C. Rackoff and B. Sloan. The Notion of Security for Probabilistic Cryptosystems. Proceedings of Crypto86, 1986.

22. Tompa, M. and H. Woll. Random Self-Reducibility and Zero-Knowledge Interactive Proofs of Possession of Information. 28th FOCS, 1986.

23. Yao, A. Theory and Applications of Trapdoor Functions. 23rd FOCS, IEEE, November, 1982, pp. 80-91.

24. Yao, A. How to Generate and Exchange Secrets. 27th FOCS, IEEE, October, 1986, pp. 162-167.