Short Discreet Proofs

Joan Boyar[1] René Peralta *[2]

[1] Department of Mathematics and Computer Science, Odense University, Campusvej 55, 5230 Odense M, Denmark (joan@imada.ou.dk)
[2] Japan Advanced Institute of Science and Technology, School of Information Science, Asahidai 15, Tatsunokuchi, Nomi, Ishikawa 923-12, Japan (peralta@cs.uwm.edu)

Abstract. We show how to produce short proofs of theorems such that a distrusting Verifier can be convinced that the theorem is true yet obtains no information about the proof itself. The proofs are non-interactive provided that the quadratic residuosity bit commitment scheme is available to the Prover and Verifier. For typical applications, the proofs are short enough to fit on a floppy disk.

1 Introduction

The main goal of this work is to show how to produce short proofs of theorems such that a distrusting Verifier can be convinced that the theorem is true yet obtains no information about the proof itself. The paper makes use of what we have called "set certification", which consists of proving that a vector of bit commitments encodes a vector of bits in a given set without revealing the bit-vector itself. We have high hopes for what useful results this technique might eventually yield. Since our shortest proofs don't exactly fit any of the published categories of zero-knowledge proofs, we have named them "discreet proofs". As an example of the power of these techniques, we show discreet proofs of knowledge of RSA and DES keys which fit on a floppy disk.

We assume that a bit commitment scheme (blobs scheme) is available as a primitive. The blobs scheme used should satisfy certain properties described in [6, 4, 5]. In addition, they must allow *non-interactive processing* of XOR gates (see [5, 3]). As a concrete example, we choose the following bit commitment scheme, which is based on the "Quadratic Residuosity Assumption" (QRA).

Definition 1. A "Blum integer" is a composite integer $N = P^r Q^s$, with P and Q primes such that $P \equiv Q \equiv 3 \ (mod\ 4)$ and r and s are odd.

* Supported in part by NSF Grant CCR-9207204. This author is currently on sabbatical leave from the University of Wisconsin at Milwaukee.

U. Maurer (Ed.): Advances in Cryptology - EUROCRYPT '96, LNCS 1070, pp. 131-142, 1996.
© Springer-Verlag Berlin Heidelberg 1996

Definition 2. (Brassard–Crépeau [7]) Let N be a fixed Blum integer produced by the Prover. "QR–blobs" are constructed as follows: the Prover selects a random $x \in Z_N^*$. To commit to a 0 the Prover sends $x^2 \bmod N$. To commit to a 1 the Prover sends $-x^2 \bmod N$.

Notation: If $x = (x_1, \ldots, x_n)$ is a vector of blobs, then $\hat{x}$ will denote the vector in GF_2^n encoded by x. Sometimes we will use $\hat{x}$ before x has been defined. In this case, x will denote any sequence of blobs which encodes $\hat{x}$.

We start by assuming that the Prover-Verifier pair has access to a common source of randomness. Our results here give non-interactive zero-knowledge proofs, in the shared random string model proposed by Blum, Feldman and Micali [1], of length $O(m \log m)$. [3] for the satisfiability of a circuit of size of size m, assuming that the security parameter is $1/m^{O(m)}$. This asymptotic complexity has also been obtained by Damgård [8], who also assumes the QRA, and impressively by Kilian and Petrank [11], who base their results on much more general assumptions. The advantage to our non-interactive proofs is in the constant factors, which are of course of practical interest. Our system beats Damgård's by a factor of about 7.5, and Petrank's and Kilian's by a very large factor.

As a next step towards our efficient discreet proofs, we allow the Prover and Verifier to engage in more than one round in which they *sequentially* query the randomness source. That is, during any round they do not have access to the random bits of the following rounds. We later remove these assumptions by simulating the randomness source. The resulting proof is efficient and non-interactive, but no longer zero-knowledge.

2 Constructible and Certifiable Sets

We start by looking at what can be done without recourse to shared randomness.

Definition 3. A subset S of GF_2^n is said to be *constructible* if it is possible for the Prover to construct, for any chosen $\hat{b} \in S$, a vector $b = (b_1, \ldots, b_n)$ of blobs, along with a proof that $\hat{b}$ belongs to S. The proof must not reveal any information about which element of S is $\hat{b}$. The proof must be non-interactive (hence we simply call it a *certificate*) and must be constructible without using the shared random string.

[3] This is not counting the necessary precomputation for establishing a blob encryption system. An efficient interactive protocol for showing that N is a Blum integer can be found in [16] A non-interactive zero-knowledge proof that N is a Blum integer can be found in [13].

For example, the existence of blob encryption schemes implies that the set $S = \{0, 1\}$ is (trivially) constructible. If the blob encryption scheme allows non-interactive processing of XOR gates then the sets $E = \{00, 11\}$ and $D = \{01, 10\}$ are also constructible. Given QR-blobs $b = (b_1, b_2)$, a proof that $\hat{b} \in E$ is simply a square root of $b_1 b_2 \bmod N$. A proof that $\hat{b} \in D$ is a square root of $-b_1 b_2 \bmod N$. For details and security proofs related to this scheme see [5].

Definition 4. A subset S of GF_2^n is said to be *certifiable* if, given any vector $b = (b_1, \ldots, b_n)$ of blobs such that $\hat{b} \in S$, the Prover can construct a proof that $\hat{b}$ belongs to S. The proof must not reveal any information about which element of S is $\hat{b}$. The proof must be non-interactive and must not use the shared random string.

The difference between certifiable and constructible sets is that, with a certifiable set, the Prover has not created the blobs herself; this gives her less freedom in producing the proof.

Note that a set which is certifiable is also constructible. Also note that the sets E and D are certifiable when QR-blobs are used. Whenever E is certifiable, it will follow that any set which is constructible is also certifiable. To see this, suppose $x = (x_1, \ldots, x_n)$ is given, and that the Prover wants to certify that $\hat{x}$ belongs to a constructible set S. To do this, the Prover simply constructs $y = (y_1, \ldots, y_n)$ such that $\hat{x} = \hat{y}$ along with a proof that $\hat{y}$ is in S. Then the Prover certifies that $\hat{x_i}\hat{y_i} \in E$ for each i. Thus, *if QR blobs are used, then certifiable sets and constructible sets are the same.*

Lemma 5. *If QR blobs are used, then any subspace S of GF_2^n is certifiable and constructible.*

Proof. Note that S must be the image of GF_2^n under multiplication by an $n \times n$ matrix M over GF_2. To certify a vector $x = (x_1, \ldots, x_n)$ of blobs, the Prover constructs blobs $y = (y_1, \ldots, y_n)$ such that $M\hat{y} = \hat{x}$. Then the Verifier and Prover can non-interactively compute blobs $v = (v_1, \ldots, v_n)$ such that $\hat{v} = M\hat{y}$ (see [3] for more details). The Prover then gives certificates that $\hat{v_i}\hat{x_i} \in E$ for each i. □

By a *linear map* from GF_2^m to GF_2^n we mean a function $f(x) = Mx + b$, where M is an $n \times m$ matrix over GF_2 and $b \in GF_2^n$. Sets of the form $f(S)$ where S is a subspace of GF_2^m will be called *cosets* (each such set is a coset of the subspace $\{v \mid v = Mx; x \in S\}$). The proof of the following lemma is analogous to the proof of Lemma 5. The details are left to the reader.

Lemma 6. *If QR blobs are used, then cosets are certifiable and constructible.*

If, for example, the set $\overline{\bigwedge} = \{001, 011, 101, 110\}$, which contains the rows of a truth table for a NAND gate, was certifiable, it would follow that a non-interactive zero-knowledge proof for circuit satisfiability is possible without using the shared random string. Unfortunately we have not been able to construct certificates for $\overline{\bigwedge}$. It may well be that, if we restrict ourselves to QR blobs, then cosets are the only certifiable sets. Other blob encryption schemes can be constructed for which sets corresponding to truth tables of various boolean functions are certifiable. But we have not been able to construct a blob encryption scheme such that a logically complete set of boolean functions is certifiable.

The next best thing to constructible/certifiable sets are sets which can be constructed/certified using the shared random string. Then the proofs that the vector defined belongs to the set would be probabilistic. Thus there would be some chance that the vector in question does not belong to the set, but that probability would be small compared to some specified security parameter.

Definition 7. A subset S of GF_2^n is said to be *probabilistically constructible* if it is possible for the Prover to construct, for any chosen $\hat{b} \in S$, a vector $b = (b_1, \ldots, b_n)$ of blobs, along with a probabilistic proof that $\hat{b}$ belongs to S. The proof must not reveal any information about which element of S is $\hat{b}$.

Definition 8. A subset S of GF_2^n is said to be *probabilistically certifiable* if, given any vector $b = (b_1, \ldots, b_n)$ of blobs such that $\hat{b} \in S$, the Prover can construct a probabilistic proof that $\hat{b}$ belongs to S. The proof must not reveal any information about which element of S is $\hat{b}$.

Both the previous constructions may use the shared random string, and the Verifier must accept an exponentially small probability that $\hat{b} \notin S$. We will show that the set $\overline{\bigwedge}$ is probabilistically certifiable. This will be used in designing non-interactive zero-knowledge proofs for circuit satisfiability, using a shared random string. It is convenient to first show that two other sets, T and U, defined below, are probabilistically certifiable.

2.1 The Set $T = \{01, 10, 11\}$

Using QR blobs, the shared random string determines, by standard techniques, a sequence $x = (x_1, \ldots, x_n)$ of randomly distributed blobs. This can be done by dividing the bits of the random string into substrings of the appropriate length k for the blobs. The substrings are interpreted as integers. The resulting integers which have Jacobi symbol $+1$ are used directly. Those with Jacobi symbol -1 are multiplied by a fixed number β with Jacobi symbol -1 (if N is constructed such that it is congruent to 5 modulo 8 then $\beta = 2$ will do). This construction requires nk shared random bits.

The number of blobs n will be specified later; it will be a function of the size of the circuit and a security parameter for the proof. With probability $1-2^{-n}$ the vector $\hat{x}$ is not the zero-vector. If the Prover chooses a random $\hat{y} \notin \{0, \hat{x}\}$ and lets $\hat{z} = \hat{x} \oplus \hat{y}$, then the set $A = \{0, \hat{x}, \hat{y}, \hat{z}\}$ is a subspace of GF_2^n. Suppose the Prover writes down two randomly chosen non-zero vectors $\hat{u}, \hat{v}$ from the set A. Note that these two vectors completely define A, since A must equal $\{0, \hat{u}, \hat{v}, \hat{u} \oplus \hat{v}\}$. If the Verifier can be convinced that $\hat{x}$ belongs to A, then this is a probabilistic proof that $\hat{x}$ is one of the three given non-zero bit vectors.

Note that A must be the kernel of a linear homomorphism $f(t) = Mt$ where M is an $(n-2) \times n$ matrix. The matrix M is easily computable non-interactively from u and v. The blob-vector z, corresponding to $M\hat{x}$ can be computed non-interactively ($z_i = \prod_{m_{i,j}=1} x_j \mod N$, where $m_{i,j}$ is the i,j-th entry in M). Now, if $M\hat{x} = 0$, then it follows that $\hat{x} \in A$. Thus, if the Prover opens all the blobs in Mx, and they are all zero, then the Verifier is convinced that x encodes one of three given non-zero bit vectors $\{\hat{x}, \hat{y}, \hat{z}\}$.

For any vector $v \in GF_2^n$, let $v^{(i)}$ denote the i^{th} bit of v. With probability larger than $1 - 3(1/2)^n$, there exist i, j such that $\{\hat{x}^{(i)}\hat{x}^{(j)}, \hat{y}^{(i)}\hat{y}^{(j)}, \hat{z}^{(i)}\hat{z}^{(j)}\} = \{01, 10, 11\}$. To see this, note that there are $4^n - 3 \cdot 2^n + 2$ possible $\hat{x}, \hat{y}$ pairs for which $\hat{x}^{(i)}\hat{y}^{(i)}$ takes at least two values from $\{01, 10, 11\}$ as i ranges from 1 to n (the third value, $\hat{z}^{(i)}$, is determined by the equation $\hat{z} = \hat{x} \oplus \hat{y}$). In the unlikely event that $\hat{x}$ is not the zero-vector yet the $\hat{x}^{(i)}\hat{y}^{(i)}$ do not take at least two values from $\{01, 10, 11\}$, the Prover simply generates another y.

Since the set $\{\hat{x}, \hat{y}, \hat{z}\}$ is known to both Prover and Verifier, both may find a pair (i, j) with the required property.[4] Then the two blobs x_i, x_j encode an element in $T = \{01, 10, 11\}$. The Verifier has no information about which element of T it is. This construction fails only when $\hat{x} = 0$. The probability of this is $1 - (1/2)^n$.

Thus we have shown how the Prover can construct two blobs which commit to a random element of the set $T = \{01, 10, 11\}$. To actually choose which two bits to commit to, the Prover can use the transformation $f(a, b) = (a \oplus b, a)$. Figure 1 shows the effect of this linear transformation.

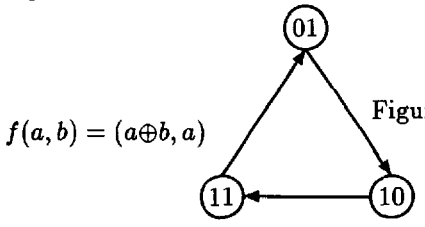

Figure 1: Going from random commitments to chosen commitments in the set T.

[4] Since typically many such pairs (i, j) will exist, some way of choosing one must be specified, e.g. choose (i, j) which minimizes $ni + j$.

To convert from a *random* member of T to a *chosen* member of T, the Prover sends two bits. The two bits tell the Verifier to advance 0, 1, or 2 positions in the diagram. i.e. the bits tell the Verifier how many times to apply the linear transformation f.

Thus we have proven

Lemma 9. *If QR blobs are used, then the set $T = \{01, 10, 11\}$ is probabilistically constructible using n random blobs with probability of failure bounded above by $(1/2)^n$. The certificate consists of $n-2$ numbers modulo N, two n-bit vectors, and two bits.*

As mentioned before, this also implies that the set T is probabilistically certifiable. But we will not make use of this result in this paper. From here on we concentrate on the techniques and cost of probabilistically constructing sets related to boolean functions. Later we also introduce techniques for reducing the amortized cost of certifying many pairs of blobs as belonging to the set T.

2.2 The Set $U = \{0111, 1011, 1101, 1110\}$

If we generate bits $ab \in T$ and $cd \in T$, then $abcd$ may take any value in

$$\{0101, 0110, 0111, 1001, 1010, 1011, 1101, 1110, 1111\}.$$

The set U may be formed by considering only the strings with odd parity. Thus the set U is probabilistically constructible as follows. Construct blobs x, y along with a proof that $\hat{x}\hat{y} \in T$. Do the same for blobs z, w. Now prove that $\hat{x} \oplus \hat{y} \oplus \hat{z} \oplus \hat{w} = 1$ by displaying a square root of $-xyzw$ modulo N. This construction guarantees that $\hat{x}\hat{y}\hat{z}\hat{w} \in U$ and, by Lemma 9, fails with probability no more than $2(1/2)^n$. Thus we have proven

Lemma 10. *If QR blobs are used, then the set $U = \{0111, 1011, 1101, 1110\}$ is probabilistically constructible using $2n$ random blobs with probability of failure bounded above by $2(1/2)^n$. The certificate consists of $2n-3$ numbers modulo N, four n-bit vectors, and four bits.*

2.3 All Two-input Boolean Gates Are Probabilistically Constructible Using QR Blobs

There are sixteen two-input boolean functions $f(x, y)$. If NAND is probabilistically constructible then all sixteen functions are probabilistically constructible. Since it will be more efficient to directly construct gates in a circuit (i.e. without simulating the gates using NAND gates), we show how to probabilistically construct the four most common gates whose truth tables do not form cosets. To do

this, use the technique of the previous section to construct blobs (a, b, c, d) which commit to a four-bit string in the set U. The following linear transformations map (a, b, c, d) to sets corresponding to boolean gates:

- $\bigwedge = \{000, 010, 100, 111\}$: $(a \oplus c, a \oplus b, b \oplus c \oplus d)$;
- $\bigvee = \{000, 011, 101, 111\}$: $(a \oplus c, a \oplus b, d)$;
- $\overline{\bigwedge} = \{001, 011, 101, 110\}$: $(a \oplus c, a \oplus b, a)$;
- $\overline{\bigvee} = \{001, 010, 100, 110\}$: $(a \oplus c, a \oplus b, a \oplus b \oplus c)$.

We leave it to the reader to verify that the image of U under each of these transformations is precisely the set of rows of the truth tables for the corresponding boolean function.

There are four more functions whose truth tables do not form cosets ($x \vee \overline{y}; \overline{x} \vee y; x \wedge \overline{y}; \overline{x} \wedge y$). We leave it to the reader to verify that appropriate transformations exist which map U to the truth tables of each of these functions. The following lemma summarizes these observations:

Lemma 11. *If QR blobs are used, then each of the subsets of GF_2^3 corresponding to the rows of the truth tables of two-input boolean functions, can be (probabilistically) constructed. For those functions which form cosets, the certificate is constructible without using the shared random strings and with zero probability of failure. For those functions which do not form cosets, the certificate is constructible using $2n$ random blobs for a probability of failure bounded above by $2(1/2)^n$. The certificate consists of $2n-3$ numbers modulo N, four n-bit vectors, and four bits.*

3 A Non-interactive Zero-knowledge Proof of Circuit Satisfiability

Let C be a circuit with m boolean gates. For simplicity we assume that all gates have 1 or 2 inputs. We assume that QR blobs are available as a primitive. That is, we will not be concerned about any precomputation necessary to establish a blob encryption system between Prover and Verifier.

Prover's algorithm.

0. The Prover commits to the values of all inputs to the circuit.
1. Use the techniques of section 2.3 to probabilistically construct commitments to the inputs and output of each non-coset gate.
2. Use the techniques of section 6 to compute the values of the outputs of coset gates.
3. For each blob x which is the output of some gate and corresponds to an input blob y of a non-coset gate, certify that $\hat{x} = \hat{y}$.

4. Open the blob corresponding to the circuit's output showing it encodes 1.

Verifier's algorithm.

1. Check each gate certification provided by the Prover in step 1 of the proof.
2. Perform step 2 of the Prover's construction just as the Prover does.
3. Verify each certification of step 3 in the proof.

The above protocol can be proven to be non-interactive zero-knowledge in the shared random string model. In the proof, the Simulator produces a shared random tape with commitments to zeros where it needs to cheat and produce something not in T. After doing that, it can get a commitment to the pair 00, claim that the pair is in T, get whatever it wants in U, and then whatever it wants for the gate for which it is cheating.

The protocol above can be extended in the obvious way to circuits with more than one output bit. In addition, although binary gates are enough to build a circuit to compute any boolean function, using gates with more than two inputs will usually reduce the number of gates in the circuit. This, in turn, will usually make the proof shorter. For example, a three-input MAJORITY gate can be simulated by 5 binary gates. However, the cost of probabilistically constructing such a gate is the same as that of probabilistically constructing a single AND gate. Here is how:

$M = \{0000, 0010, 0100, 0111, 1000, 1011, 1101, 1111\}$ is the set that must be constructed. Recall the set U from section 2.2. The set $V = \{abcde \mid abcd \in U; e \in GF_2\}$ is clearly constructible at the same cost as constructing an element from U. Now note that the transformation $abcde \rightarrow (a \oplus c \oplus e, a \oplus b \oplus e, e, b \oplus c \oplus d \oplus e)$ maps V onto M.

Note that step 3 of both the Verifier's and the Prover's algorithm is simply to check that $\hat{x} \oplus \hat{y}$ is the zero vector for many pairs x, y. In sections 4 and 5 we will introduce techniques that allow the Prover to simply skip step 3 and the Verifier to exchange step 3 by a probabilistic constant-cost test. An analogous modification to step 1 will drastically reduce the length of the proof.

4 A Two-round Discreet Proof of Circuit Satisfiability

In lemma 11, we show that constructing input-output blobs for any boolean gate (with probability of error less than $2(1/2)^n$) can be done by opening about $2n$ blobs and sending about $4n$ bits. The purpose of opening the $2n$ blobs is to show that they are all 0. Step 3 of the Prover's and Verifier's algorithm also amounts to showing that a number of blobs are 0. Rather than opening these blobs the Prover can *probabilistically* show that all blobs are 0 as follows: If not all blobs are 0, then the exclusive-or of a random subset of the blobs is 1 with probability

1/2. Thus the Verifier can be convinced that all blobs are 0, with confidence level $1-(1/2)^n$, by the Prover opening n non-interactively computed blobs. However, one more query to the randomness source is needed to select n random subsets of the blobs (alternatively, the Verifier could select the random subsets). Thus, if we allow two rounds in our proof, then the Prover need only send about $4n$ m bits plus n blob openings to prove circuit satisfiability with exponentially small probability of getting away with a false proof.[5] This is an extremely short zero-knowledge proof. In the next section we see how these observations allow us to construct practical non-interactive proofs. We also postpone, until section 5.1, the discussion of the actual error probability as a function of n and m.

5 How to Construct Non-interactive Discreet Proofs by Simulating the Randomness Source

It is a standard technique in cryptographic protocol design to substitute a random "challenge string" by a string constructed deterministically but in such a way that the Prover has no control. For example, Fiat-Shamir's scheme [10], Schnorr's scheme [15], and the DSA all use a one-way hash function to construct a "challenge" to the Prover, hence eliminating the need for interaction.

The proofs described in this work consist of rounds where Prover and Verifier first obtain a random string from a trusted source and then the Prover sends a message to the Verifier. We can replace the random string by the output of a cryptographically secure pseudo-random number generator. For the first round we seed the generator with the description of the circuit. For the following rounds[6] we seed the generator with the Prover's message in the previous round. The generator must be run in the forward direction (so that both the Prover and the Verifier can compute the output without interaction). Note that the simulation of the randomness source converts the two-round protocol into a one-round (i.e. non-interactive) proof. Note also that the random strings are not part of the proof. Considering the security parameter a constant (or even $1/m^{O(m)}$), the simulation yields a non-interactive proof of size $O(m \log m)$ bits (see section 5.1 for the explanation of the $\log(m)$ factor). The theoretical justification is, of course, not as clean as that for non-interactive zero-knowledge proofs as introduced by Blum et. al. (see [2, 9]). However, our proofs are short enough to be used in practice (e.g. they will typically fit on a floppy disk). A more precise analysis of the length and the error probability follows.

[5] The error probability is slightly more than $1-(1/2)^n$. We will be more precise about this in section 5.

[6] We simulate only two rounds in this paper, but the technique is applicable to any number of rounds.

There is a temptation to apply our methods to earlier interactive zero-knowledge proofs for circuit satisfiability (such as [7] [5],[3]) in order to obtain even more efficient non-interactive proofs. This fails because in those protocols, the Verifier's challenges are too short when the security parameter is $1/m^{O(m)}$. This means that while producing the non-interactive proof, a cheating Prover can simply try random values until it finds one that satisfies all the challenges.

5.1 Performance

Until now we have been assuming the worse case scenario: that all gates in the circuit are non-coset gates. This is clearly unrealistic. Therefore we introduce a new parameter θ to denote the number of non-coset gates of the circuit (The tired reader may choose to think of the circuit as containing only AND, XOR, and NOT gates. In this case θ is just the number of AND gates).

Recall that k is the size of the Blum integer N. The length of the proof is essentially $n(4\theta + k)$ where n is a security parameter which determines the probability that the Prover can get away with a false proof.

There are two events which *might* allow the Prover to cheat:[7]

1. In the process of constructing the commitments to a gate, a vector $\hat{x}$ derived from the random source is the zero vector.
2. In the final step of the proof not all blobs are 0, yet the exclusive-or of the n random subsets of the blobs are all 0.

A moment's thought will convince the reader that the probability of the first of these events dominates for moderately large values of θ. Therefore we concentrate on this event only. There are 4θ vectors to worry about. The probability that at least one of these vectors is 0 is $1-(1-(1/2)^n)^{4\theta}$. Letting $n = \log_2(\theta)+r$, we have that $1-(1-(1/2)^n)^{4\theta} \longrightarrow 2^{2-r}$.

In practice, we can take $r = 50$. The convergence is fast. Thus the length of the proof, in bits, is

$$(4\theta + k)(\log_2(\theta) + 50) = 200\,\theta + 4\theta \log_2(\theta) + k \log_2(\theta) + 50\,k. \qquad (I)$$

Recent developments in factorization suggest we best take $k = 1024$ (two succinct and up-to-date discussions on the status of factorization algorithms and their implementations are contained in [12, 14]). Nevertheless, the term $200\,\theta$ dominates this expression for practical applications.

[7] In practice even if one of these events occur, the Prover is unlikely to be able to cheat.

6 Length of Discreet Proofs For RSA and DES

The aim of this section is to show that our proofs are short enough to be used in practice for commonly used cryptographic functions. No attempt at circuit optimization has been done. That is the subject of work in progress. We note that the construction of circuits which minimize the number of non-coset gates is a new problem. Until now, circuit designers would have had no reason to consider such a problem.

Suppose we want to prove to the world that we know a DES key K such that $\mathrm{DES}_K(X) = Y$ for public X and Y. The techniques introduced in this paper involve constructing a circuit for DES where the unknown input is the key K. The circuit must be constructed so as to minimize the number of AND gates. In DES, AND gates are only needed for indexing into the S-boxes. A straightforward construction, without exploiting any structure in the S-boxes (there shouldn't be any ☺), yields a circuit with 57 AND gates per S-box. There are 8 S-boxes and each is used 16 times. Therefore the number of AND gates in our circuit is 7296. This is the value of θ defined in the previous section. Equation (I) then yields approximately 240 kilobytes.

Let us now consider proving to the world that we know and RSA decryption key d. Since knowing d is poly-time equivalent to knowing the factorization of N, we can prove this fact instead. Therefore all we need is a circuit which multiplies two inputs P and Q, verifies that the product is equal to a hard-wired N, and verifies that P is not 1 or N.

Assume N is of length $2\,d$ bits. Standard techniques yield a circuit of size about $d^{1.6}$. For $d = 512$, equation (I) yields approximately 700 kilobytes.

References

1. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications. In Proceedings of the 20th ACM Symposium on Theory of Computing (1988) pp. 103-112.
2. Blum, De Santis, Micali, Persiano: Non-interactive zero-knowledge. SIAM Journal on Computing **20** (1991) 1084–1118.
3. Boyar, J., Brassard, G., Peralta, R.: Subquadratic zero-knowledge. In Proceedings of the 32th IEEE Symposium on Foundations of Computer Science (1991) pp. 69–78 (to appear in JACM).
4. Boyar, J., Krentel, M., Kurtz, S.: A discrete logarithm implementation of zero-knowledge blobs. Journal of Cryptology **2** (1990) 63–76.
5. Boyar, J., Lund, C., Peralta, R.: On the communication complexity of zero-knowledge proofs. Journal of Cryptology **6** (1993) 65–85.

6. Brassard, G., Chaum, D., Crépeau, C.: Minimum disclosure proofs of knowledge. Journal of Computer and System Sciences **37** (1988) 156–189.
7. Brassard, G., Crépeau, C.: Zero-knowledge simulation of boolean circuits. In Advances in Cryptology - Proceedings of CRYPTO 86 (1987) vol. 263 of Lecture Notes in Computer Science, Springer-Verlag pp. 223–233.
8. Damgård, I.: Non-interactive circuit based proofs and non-interactive perfect zero-knowledge with preprocessing. In Advances in Cryptology - Proceedings of EUROCRYPT 92 (1993) vol. 658 of Lecture Notes in Computer Science, Springer-Verlag pp. 341–355.
9. Feige, U., Lapidot, D., Shamir, A.: Multiple non-interactive zero-knowledge proofs based on a single random string. In Proceedings of the 31th IEEE Symposium on Foundations of Computer Science (1990) pp. 308–317.
10. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In Advances in Cryptology - Proceedings of CRYPTO 86 (1987) vol. 263 of Lecture Notes in Computer Science, Springer-Verlag pp. 186–194.
11. Kilian, J., Petrank, E.: An efficient non-interactive zero-knowledge proof ystem for NP with general assumptions. Technical Report No. TR95-038, Electronic Colloquium in Computational Complexity (ECCC), July 1995.
12. Lenstra, A. K.: Factoring integers using the web and number field sieve. In Proceedings of JAIST International Forum on Multimedia and Information Security (1995) Japan Advanced Institute of Science and Technology pp. 93–113.
13. De Santis, A., Di Crescenzo, G., Persiano, G.: Secret sharing and perfect zero knowledge. In Advances in Cryptology - Proceedings of CRYPTO 93 (1993) vol. 773 of Lecture Notes in Computer Science, Springer-Verlag pp. 73–84.
14. Odlyzko, A.: The future of integer factorization. In Proceedings of JAIST International Forum on Multimedia and Information Security (1995) Japan Advanced Institute of Science and Technology pp. 139–151.
15. Schnorr, C.: Efficient signature generation for smart cards. Journal of Cryptology **4** (1991) 161–174.
16. van de Graaf, J., Peralta, R.: A simple and secure way to show the validity of your public key. In Advances in Cryptology - Proceedings of CRYPTO 87 (1988) vol. 293 of Lecture Notes in Computer Science, Springer-Verlag pp. 128–134.