J. Cryptology (2002) 15: 251–269
DOI: 10.1007/s00145-002-0112-1

Journal of
CRYPTOLOGY

Paillier's Trapdoor Function Hides up to $O(n)$ Bits*

Dario Catalano
Département d'Informatique, Ecole Normale Supérieure,
45 Rue d'Ulm, 75230 Paris, France
Dario.Catalano@ens.fr

Rosario Gennaro
I.B.M. T. J. Watson Research Center,
P.O. Box 704, Yorktown Heights, NY 10598, U.S.A.
rosario@watson.ibm.com

Nick Howgrave-Graham
NTRU Cryptosystems, 5 Burlington Woods,
Burlington, MA 01803, U.S.A.
NHowgraveGraham@ntru.com

Communicated by Matt Franklin

Received May 2001 and revised March 2002
Online publication 12 August 2002

Abstract. At EuroCrypt '99 Paillier proposed a new encryption scheme based on higher residuosity classes. The new scheme was proven to be one-way under the assumption that *computing* N-residuosity classes in $Z^*_{N^2}$ is hard. Similarly the scheme can be proven to be semantically secure under a much stronger *decisional* assumption: given $w \in Z^*_{N^2}$ it is impossible to decide if w is an N-residue or not.

In this paper we examine the bit security of Paillier's scheme. We prove that if computing residuosity classes is hard, then given a random w it is impossible to predict the least significant bit of its class significantly better than at random. This immediately yields a way to obtain semantic security without relying on the decisional assumption (at the cost of several invocations of Paillier's original function).

In order to improve efficiency we then turn to the problem of simultaneous security of many bits. We prove that Paillier's scheme hides $n - b$ (up to $O(n)$) bits if one assumes that computing the class c of a random w remains hard even when we are told that $c < 2^b$. We thoroughly examine the security of this stronger version of the intractability of the class problem.

An important theoretical implication of our result is the construction of the first trapdoor function that hides super-logarithmically (up to $O(n)$) many bits. We generalize our techniques to provide sufficient conditions for a trapdoor function to have this property.

Key words. Trapdoor functions, Bit security, Semantic security.

* A preliminary version of this work appears in the *Proceedings of Eurocrypt* 2001. Work by the first author was done while visiting the Computer Science Department of Columbia University, supported by the Dipartimento di Matematica e Informatica, Università di Catania, Italy. Work by the third author was done while at the IBM T.J. Watson Research Center.

1. Introduction

At EuroCrypt '99 Paillier [13] proposed a new encryption scheme based on higher residuosity classes. It generalized previous work by Okamoto and Uchiyama [12]. Both works are based on the problem of computing high-degree residuosity classes modulo a composite of a special form (in [13] the modulus is N^2 where N is a typical RSA modulus, while in [12] the modulus is $N = p^2q$ where p, q are large primes).

The mathematical details are described below, but for now we sketch the basics of Paillier's scheme. It can be shown that $Z^*_{N^2}$ can be partitioned into N equivalence classes generated by the following equivalence relationship: $a, b \in Z^*_{N^2}$ are equivalent iff ab^{-1} is an N-residue in $Z^*_{N^2}$. The N-residuosity class of $w \in Z^*_{N^2}$ is the integer $c = Class(w)$ such that w belongs to the cth residuosity class (in a well specified ordering of them). The conjectured hard problem is: given a random w, compute c. It can be shown that computing $c = Class(w)$ is possible if the factorization of N is known.

Thus Paillier suggests the following encryption scheme: To encrypt a message $m \in Z_N$, the sender sends a random element $w \in Z^*_{N^2}$ such that $Class(w) = m$ (this can be done efficiently as is shown later). The receiver who knows the factorization of N, given w can compute m.

If we assume that computing residuosity classes is hard, then this scheme is simply one-way. Indeed, even if computing the whole of m is hard, it is possible that partial information about m can be leaked.

What we would like to have is instead a *semantically secure* scheme. Semantic security (introduced by Goldwasser and Micali in [10]) basically says that to a polynomial time observer the encryption of a message m should look indistinguishable from the encryption of a different message m'. Paillier's scheme is semantically secure if we assume a stronger *decisional* assumption: given a random element $w \in Z^*_{N^2}$ it is impossible to decide efficiently if w is an N-residue or not.

Hard-Core Bits. The concept of hard-core bits for one-way functions was introduced by Blum and Micali in [4].

Given a one-way function $f\colon \{0,1\}^n \to \{0,1\}^n$ we say that $\pi\colon \{0,1\}^n \to \{0,1\}$ is a hard-core predicate for f if given $y = f(x)$ it is hard to guess $\pi(x)$ with probability significantly higher than $1/2$. Another way of saying this is that if x is chosen at random, then $\pi(x)$ looks random (to a polynomial time observer) even when given $y = f(x)$.

Blum and Micali [4] showed the existence of a hard-core predicate for the discrete logarithm function. Later a hard-core bit for the RSA/Rabin functions was presented by Alexi et al. [1]. Goldreich and Levin [8] show that any one-way function has a hard-core predicate.

The concept can be generalized to many hard bits. We say that k predicates $\pi_1, \ldots, \pi_k$ are *simultaneously* hard-core for f if given $f(x)$ the collection of bits $\pi_1(x), \ldots, \pi_k(x)$ looks random to a polynomial time observer.

Our Result. In this paper we investigate the hard core bits of Paillier's new trapdoor scheme. We first prove that the least significant bit of $c = Class(w)$ is a hard-core bit if we assume computing residuosity classes is hard. In other words we show that given a

random $w \in Z^*_{N^2}$, if one can guess $lsb(Class(w))$ better than at random, then one can compute the whole $Class(w)$ efficiently.

Let $n = |N|$. The result above can be generalized to the simultaneous hardness of the least $O(\log n)$ bits using standard techniques. We then show that by slightly strengthening the assumption on computing residuosity classes we are able to extract many more simultaneously hard-core bits. More precisely, for any $\omega(\log n) \leq b < n$ we show that Paillier's scheme hides the $n - b$ least significant bits, if we assume that computing residuosity classes remains hard even if we are told that the class is smaller than 2^b.

The residuosity class problem seems to remain hard even in this case. Actually we see *no* way to exploit knowledge of the bound (i.e. the fastest known algorithm to compute c even in this case is to factor N). We discuss this further in Section 3.4.

An interesting feature of our construction is that the number of bits hidden by the functions is related to the underlying complexity assumption that one is willing to make. The smaller the bound is (i.e. the stronger the assumption), the more bits one can hide.

A Theoretical Implication. If f is a trapdoor permutation that simultaneously hides k bits, then we can securely encrypt k bits with a single invocation of f (as originally described by Goldwasser and Micali [10]).

However, for all previously known trapdoor functions (like RSA) $f\colon \{0,1\}^n \to \{0,1\}^n$ we only know how to prove that $k = O(\log n)$ bits are simultaneously hard-core. Thus to securely encrypt m bits one needs to invoke the function $\Omega(m/\log n)$ times.

Another way to look at our result is that we show a candidate trapdoor function that hides up to $O(n)$ bits. To our knowledge this is the first example of a trapdoor problem with a super-logarithmic number of hard-core predicates.

We also generalize our constructions to a large class of trapdoor functions by giving sufficient conditions for a trapdoor function to hide super-logarithmically many bits.[1]

Decisional Assumptions. As we mentioned earlier, the scheme of Paillier [13] can also be proven to be semantically secure under a decisional problem involving residuosity classes. In other words if assuming that deciding N-residuosity is hard, then these schemes hide *all* n input bits.

Notice however the difference with our result. We prove that the scheme hides many bits, under a *computational* assumption related to computing residuosity class.

Decisional assumptions are very strong. Basically a decisional problem is a true/false question which we assume the adversary is not able to solve. Conversely computational assumptions require the adversary to compute the full solution of a computational problem (similar to a free-response question). As any college student will tell you, a test composed of true/false questions is much more likely to be easier than a test composed of free-response questions. Thus, whenever possible, computational assumptions should be preferred to decisional ones.

[1] The above discussion implicitly rules out *iterated* functions. Indeed Blum and Micali [4] show that if $f(x)$ is a one-way function and $\pi(x)$ is a hard-core predicate for it, then the iterated function $f^k(x)$ is clearly also one-way and it simultaneously hides the following k bits: $\pi(x), \pi(f(x)), \ldots, \pi(f^{k-1}(x))$. We are interested in functions that hide several bits in a *single* invocation.

This is however an intuitive statement. Regarding our particular problem, it should be pointed out that we do not know of any reductions beyond the obvious one that computing the N-residuosity class implies deciding N-residuosity. We do not know any implication between the restricted assumption we make (i.e. computing the class when we are told it is a b-bit number) and deciding N-residuosity.

The goal of this paper is nevertheless to show example of trapdoor functions that hides several bits without resorting to true/false questions.

Applications. The main application of our result is the construction of a new semantically secure encryption scheme based on Paillier's scheme. Assuming that Paillier's function securely hides k bits, we can then securely encrypt an m-bit message using only $O(m/k)$ invocations; k is of course a function of n, the security parameter of the trapdoor function. We can do this without resorting to the decisional assumption about N-residuosity, but simply basing our security on the hardness of computing residuosity classes.

Today we can assume that $n = 1024$. Also in practice public-key cryptography is used to exchange keys for symmetric encryption. Thus we can then assume that $m = 128$. With a reasonable computational assumption we can encrypt the whole 128-bit key with a *single* invocation of Paillier's scheme. The assumption is that computing the class is hard even when we are promised that $c < N^{0.875}$.

We discuss this new scheme and make comparisons with existing ones in Section 5.

1.1. *Related Work*

Computing high-degree residuosity classes is related to the original work of Goldwasser and Micali [10] who suggested quadratic residuosity in Z_N^* as a hard trapdoor problem (where N is an RSA modulus). Later Benaloh [2] generalized this to deciding s-residuosity where s is a small prime dividing $\varphi(N)$. In Benaloh's scheme, s is required to be small (i.e. $|s| = O(\log n)$) since the decryption procedure is exponential in s. By changing the structure of the underlying field, Okamoto and Uchiyama [12] and Paillier [13] were able to lift this restriction and consider higher-degree residuosity classes. More recently Catalano et al. [6] show how to modify Paillier's scheme in order to improve its efficiency.

The idea of restricting the size of the input space of a one-way function in order to extract more hard bits goes back to Hastad et al. [11]. They basically show that the ability to invert $f(x) = g^x \bmod N$ when x is a random integer $x < O(\sqrt{N})$ is sufficient to factor N. Then they show that discrete log modulo a composite must have $n/2$ simultaneously hard bits, otherwise the above restricted-input function can be inverted (i.e. we could factor N). Thus [11] shows the first example of a one-way function with a super-logarithmic number of hard-core bits. No such examples were known for *trapdoor* functions.

Building on ideas from Hastad et al. [11], Patel and Sundaram [14] show that if one assumes that $f(x) = g^x \bmod p$ (with p prime) remains hard to invert even when $x < B$, then the discrete logarithm simultaneously hides $k - b$ bits ($k = |p|, b = |B|$). In their case, as in ours, one must make an explicit computational assumption about the hardness of inverting the function with small inputs. There is an important difference between

the Patel–Sundaram assumption [14] and ours. For the case of the discrete logarithm function, we know that there exist algorithms to find $x < B$ given $y = g^x$, which run in $O(\sqrt{B})$ steps. In our case, as discussed in Section 3.4, an attack with a similar complexity is not known.

1.2. *Paper Organization*

In Section 2 we provide some basic definitions. In Section 3 we describe in detail the scheme based on Paillier's function. In Section 4 we generalize our result to a larger class of trapdoor functions, giving sufficient conditions for a trapdoor function to hide super-logarithmically many bits. We then discuss applications to public-key encryption and comparisons with other schemes in Section 5. Our work raises some interesting open problems which we list at the end in Section 6.

2. Definitions

In the following we denote with $\mathbf{N}$ the set of natural numbers and with $\mathbf{R}^+$ the set of positive real numbers. We say that a function $\mathsf{negl}\colon \mathbf{N} \to \mathbf{R}^+$ is *negligible* iff for every polynomial $P(n)$ there exists an $n_0 \in \mathbf{N}$ s.t. for all $n > n_0$, $\mathsf{negl}(n) \leq 1/P(n)$. We denote with $\mathcal{PRIMES}(k)$ the set of primes of length k. For $a, b \in \mathbf{N}$ we write $a \propto b$ if a is a non-zero multiple of b.

If A is a set, then $a \leftarrow A$ indicates the process of selecting a at random and uniformly over A (which in particular assumes that A can be sampled efficiently).

Trapdoor Permutations (from [9]). Let I be a set of indices. A family of *one-way trapdoor permutations* is a set $F = \{f_n\colon \{0,1\}^n \to \{0,1\}^n\}_{n\in I}$ satisfying the following conditions:

- There exists a polynomial p and a probabilistic polynomial time Turing Machine S_1 which on input 1^k (where k is a security parameter) outputs pairs (n, t_n) where $n \in I \cap \{0,1\}^k$ and $|t_n| < p(k)$. The information t_n is referred to as the trapdoor.
- There exists a probabilistic polynomial time Turing Machine S_2 which on input $n \in I$ outputs $x \in \{0,1\}^n$.
- There exists a probabilistic polynomial time Turing Machine A_1 such that for $n \in I$, $x \in \{0,1\}^n$, $A_1(n, x) = f_n(x)$.
- There exists a probabilistic polynomial time Turing Machine A_2 such that $A_2(n, t_n, f_n(x)) = x$, for all $x \in \{0,1\}^n$ and for all $n \in I$.
- For every probabilistic polynomial time Turing Machine $\mathcal{A}$ we have that, for large enough k,

$$\Pr[f_n(x) = y\colon\ n \leftarrow I; x \leftarrow \{0,1\}^n; \mathcal{A}(n, y) = x] = \mathsf{negl}(k).$$

The above notion can be generalized to *probabilistic* functions where each $f_n\colon \{0,1\}^n \times \{0,1\}^r \to \{0,1\}^{n+r}$ is a permutation, but we look at the second argument as a random string and we assume that given $y \in \{0,1\}^{n+r}$ we cannot compute the first argument, i.e.

for any probabilistic polynomial time Turing Machine $\mathcal{A}$ we have that

$$\Pr[x \leftarrow \{0,1\}^n; s \leftarrow \{0,1\}^r; \mathcal{A}(f_n, f_n(x,s)) = x] = \mathsf{negl}(n).$$

Hard-Core Bits. A Boolean predicate π is said to be *hard* for a function f_n if no efficient algorithm $\mathcal{A}$, given $y = f_n(x)$, guesses $\pi(x)$ with probability substantially better than $1/2$. More formally for any probabilistic polynomial time Turing Machine $\mathcal{A}$ we have that

$$|\Pr[x \leftarrow \{0,1\}^n; \mathcal{A}(f_n, f_n(x)) = \pi(x)] - \tfrac{1}{2}| = \mathsf{negl}(n).$$

For one-way functions f_n, a possible way to prove that a predicate π is hard is to show that an efficient algorithm $\mathcal{A}$ that on input $y = f_n(x)$ guesses $\pi(x)$ with probability bounded away from $1/2$ can be used to build another algorithm $\mathcal{A}'$, running in probabilistic polynomial time, that on input y computes x with non-negligible probability.

Simultaneously Hard Bits. A collection of k predicates $\pi_1, \ldots, \pi_k$ is called simultaneously hard-core for f_n if, given $y = f_n(x)$, the whole collection of bits $\pi_1(x), \ldots, \pi_k(x)$ looks "random." A way to formalize this (following the work of Yao [17]) is to say that it is not possible to guess the value of the jth predicate even after seeing $f_n(x)$ and the value of the previous $j-1$ predicates over x. Formally, for every $j = 1, \ldots, k$, for every probabilistic polynomial time Turing Machine $\mathcal{A}$ we have that

$$|\Pr[x \leftarrow \{0,1\}^n; \mathcal{A}(f_n, f_n(x), \pi_1(x), \ldots, \pi_{j-1}(x)) = \pi_j(x)] - \tfrac{1}{2}| = \mathsf{negl}(n).$$

Here, too, a proof method for simultaneously hard-core bits is to show that an efficient algorithm $\mathcal{A}$ contradicting the above equation can be used to build another efficient algorithm $\mathcal{A}$ which inverts f_n with non-negligible probability.

3. Bit Security of Paillier's Scheme

In this section we present our candidate trapdoor function which is based on work by Paillier [13]. Readers are referred to Paillier's paper [13] for details and proofs which are not given here.

Preliminaries. Let $N = pq$ be an RSA modulus, i.e. the product of two large primes of roughly the same size. Consider the multiplicative group $Z^*_{N^2}$.

Let $g \in Z^*_{N^2}$ be an element whose order is a non-zero multiple of N. We denote with $\mathcal{B}$ the set of such elements. It can be shown that g induces a bijection

$$\mathcal{E}_g\colon Z_N \times Z^*_N \to Z^*_{N^2},$$
$$\mathcal{E}_g(x,y) = g^x y^N \bmod N^2.$$

Thus, given g, for an element $w \in Z^*_{N^2}$ there exists a unique pair $(c,z) \in Z_N \times Z^*_N$ such that $w = g^c z^N \bmod N^2$. We say that c is the *class* of w relative to g. We may also denote this with $Class_g(w)$.

We define the *Computational Composite Residuosity Class Problem* as the problem of computing c given w and assume that it is hard to solve.

Definition 1. We say that computing the function $Class_g(\cdot)$ is hard if, for every probabilistic polynomial time algorithm $\mathcal{A}$, there exists a negligible function $\mathsf{negl}()$ such that

$$\Pr\begin{bmatrix} p, q \leftarrow \mathcal{PRIMES}(n/2); \quad N = pq; \\ g \leftarrow Z^*_{N^2} \text{ s.t. } ord(g) \propto N; \\ c \leftarrow Z_N; \quad z \leftarrow Z^*_N; \quad w = g^c z^N \bmod N^2; \\ \mathcal{A}(N, g, w) = c \end{bmatrix} = \mathsf{negl}(n).$$

It can be shown that if the factorization of N is known, then one could solve this problem: indeed, let $\lambda = \lambda(N) = \text{lcm}(p-1, q-1)$, where $\lambda(\cdot)$ is the *Carmichael* function,[2] then

$$Class_g(w) = \frac{L(w^\lambda \bmod N^2)}{L(g^\lambda \bmod N^2)} \bmod N, \tag{1}$$

where L is defined as the integer[3] $L(u) = (u-1)/N$. On the other hand, if the factorization is not known, no polynomial strategy to solve the problem has, so far, been discovered. This leads to the following conjecture:

Assumption 1. *If N is a modulus of unknown factorization, there exists no probabilistic polynomial time algorithm for the Computational Composite Residuosity Class Problem.*

An interesting property of the class function is that it is homomorphic: for $x, y \in Z^*_{N^2}$,

$$Class_g(xy \bmod N^2) = Class_g(x) + Class_g(y) \bmod N.$$

It is also easy to see that $Class_g(\cdot)$ induces an equivalence relationship (where elements are equivalent if they have the same class) and thus for each c we have $\varphi(N)$ elements in $Z^*_{N^2}$ with class equal to c.

3.1. *The Least Significant Bit of Class is Hard*

As we said in the Introduction, Goldreich and Levin [8] proved that any one-way function has a hard-core bit. Clearly their result applies to Paillier's scheme as well. Here, however, we present a direct and more efficient construction of a hard-core bit.

Consider the function $Class_g(\cdot)$ defined as in the previous section. Then, given $w = g^c y^N \bmod N^2$, for some $c \in Z_N$ and $y \in Z^*_N$, computing the predicate $lsb(c)$ is equivalent to computing $Class_g(w)$, i.e. $lsb(c)$ is hard for $Class_g$. We start with the following lemma.

---

[2] Note that the Carmichael function, on input an integer t, is defined as follows:

If $t = 2^l$ for some l,

$$\lambda(t) = \begin{cases} 2^{l-1} & \text{if } \ l < 3, \\ 2^{l-2} & \text{if } \ l \geq 3. \end{cases}$$

If $t = 2^{e_0} p_1^{e_1} \cdots p_k^{e_k}$,

$$\lambda(t) = \text{lcm}(\lambda(2^{e_0}, \varphi(p_1^{e_1}), \ldots, \varphi(p_k^{e_k}))).$$

This implies that when N is an RSA modulus, $\lambda(N) = \text{lcm}(p-1, q-1)$.

[3] It is easy to see that both w^λ and g^λ are $\equiv 1 \bmod N$.

Lemma 1. *Let N be a random n-bit RSA modulus, let $y \in Z_N^*$, let c be an even element of Z_N and let g be an element in $\mathcal{B}$. Then, denoting $z = 2^{-1} \bmod N$,*

$$(g^c y^N)^z = g^{c/2} y'^N \bmod N^2$$

for some $y' \in Z_N^*$

Proof. Since $z = 2^{-1} \bmod N$, there exists an integer k such that $2z = 1 + kN$. Now

$$(g^c y^N)^z = g^{2z(c/2)} y^{zN} \bmod N^2 = g^{c/2}(g^{ck/2} y^z)^N \bmod N^2.$$

Since the group Z_{N^2} is isomorphic to $Z_N^* \times Z_N$ (for $g \in \mathcal{B}$) [13], this would already be enough to conclude the proof. However, for completeness, we provide here a more precise argument.

It remains to show that $(g^{ck/2} y^z)^N = y'^N \bmod N^2$ for some $y' \in Z_N^*$. Observe that $(g^{ck/2} y^z)$, being an element of $Z_{N^2}^*$, can be written as $a + bN$ with $a \in Z_N^*$ and $b \in \{0, \ldots, N\}$. This implies that by putting $y' = a$ the above equation is satisfied. This completes the proof. □

Informally, Lemma 1 gives us an easy way to perform "bit shifts" to the right, meaning with this the following operation. Assume we have an element $w = g^c y^N$ (where $c = c'0$ is an even integer), by applying the lemma we get the value $w' = g^{c'} y'^N \bmod N^2$. Note that $Class_g(w')$ is obtained from $Class_g(w)$ by simply eliminating its least significant bit and by "moving" all the remaining bits one position to the right.

Theorem 1. *Let N be a random n-bit RSA modulus, and let the functions $\mathcal{E}_g(\cdot, \cdot)$ and $Class_g(\cdot)$ be defined as above. If the function $Class_g(\cdot)$ is hard (see Definition 1), then the predicate $lsb(\cdot)$ is hard for it.*

Proof. The proof goes by reductio ad absurdum: we suppose the given predicate not to be hard, and then we prove that if some oracle $\mathcal{O}$ for $lsb(\cdot)$ exists, then this oracle can be used to construct an algorithm that computes the assumed intractable function, in probabilistic polynomial time. In other words, given $w \in Z_{N^2}^*$ such that $w = \mathcal{E}_g(c, y)$, and an oracle $\mathcal{O}(g, w) = lsb(c)$, we show how to compute, in probabilistic polynomial time, the whole value $c = Class_g(w)$.

For the sake of clarity we divide the proof in two cases, depending on what kind of oracle is given to us. In the first case we suppose we have access to a perfect oracle, that is an oracle for which $\Pr_w[\mathcal{O}(g, w) = lsb(c)] = 1$. Then we will show how to generalize the proof for the more general case in which the oracle is not perfect, but has some non-negligible advantage in predicting the required bit. In this last case we suppose $\Pr_w[\mathcal{O}(g, w) = lsb(c)] \geq \frac{1}{2} + \varepsilon(n)$ where $\varepsilon(n) > 1/p(n)$, for some polynomial $p(\cdot)$. For convenience we denote $\varepsilon(n)$ simply by ε in the following analysis.

The Perfect Case. The algorithm computes c, bit by bit starting from $lsb(c)$. Denote by $c = c_n \cdots c_2 c_1$ the bit expansion of c. It starts by querying $\mathcal{O}(g, w)$ which, by assumption, will return $c_1 = lsb(c)$. Once we know c_1 we can "zero it out" by using the homomorphic

Compute Class Algorithm (I)

ComputeClass$(\mathcal{O}, w, g, N)$

1. $z = 2^{-1} \bmod N$
2. $c = ()$
3. **for** $i = 0$ **to** $n = |N|$
4. $\quad x = \mathcal{O}(g, w)$
5. $\quad$ **prepend**(x, c)
6. $\quad$ **if** (x==1) **then**
7. $\quad\quad w = w \cdot g^{-1} \bmod N^2$ (bit zeroing)
8. $\quad w = w^z \bmod N^2$ (bit shifting)
9. **return** c

Fig. 1. Pseudocode description of the algorithm for the perfect oracle case.

properties of the function *Class*. This is done by computing $w' = w \cdot g^{-c_1}$. Finally we use Lemma 1 to perform a "bit shift" and position c_2 in the *lsb* position. We then iterate the above procedure to compute all of c. A detailed description of the algorithm is given in Fig. 1 (where () denotes the empty string and **prepend**(x, c) prepends the string/bit x to the string c).

The Imperfect Oracle. In this case the above algorithm does not work, because we are not guaranteed that x is the correct bit during any of the iterations. We need to use randomization to make use of the statistical advantage of the oracle in guessing the bit. This is done by considering $\hat{w} = w \cdot g^r \cdot s^N$ and querying $\mathcal{O}(g, \hat{w})$ on several randomized $\hat{w}$'s. The number l of queries is a function of the advantage $\varepsilon(n)$ of the oracle (recall that we are assuming that $\varepsilon^{-1}(n)$ is bounded by a polynomial in n).

Notice that if $c + r < N$ the oracle returns as output $c_1 + r_1 \bmod 2$, and since we know r_1 we can compute c_1. A majority vote on the result of all the queries will be the correct c_1 with very high probability.

In order to ensure that $c + r < N$, we somewhat "reduce" the size of c. We guess the top $\gamma = 1 - \log \varepsilon$ bits of c, and zero them accordingly, i.e.

$$w'_d = g^{2^{n-\gamma} d} w$$

for all 2^γ choices of d (note that is is a polynomial, in n, number of choices).

Of course if we guessed incorrectly the actual top bits of w'_d will not be zeroed, however for one of our guesses they will be, and this guess will yield the correct answer.

Observe that since we zeroed the leading γ bits of c, the sum $r + c$ can wrap around N only if the γ most significant bits of r are all 1. Thus the probability of $r + c > N$ is smaller that $2^{-\gamma} = \varepsilon/2$. We can add this probability to the error probability of the oracle. Consequently the oracle is now correct with probability $1/2 + \varepsilon/2$. This simply implies that we need to increase the number of randomized queries accordingly.

Once c_1 is known, we zero it and we perform a shift to the right as before. We then repeat the process for the remaining bits. Since the correct d is still unknown, we only obtain a (polynomially sized) set of candidate values for c. However, this still implies an algorithm to output c correctly with non-negligible probability, which contradicts Definition 1.

A complete description of the algorithm is presented in Fig. 2.

Compute Class Algorithm (II)

ComputeClass′$(\mathcal{O}, w, g, N, \varepsilon)$

1. $\delta = 1/2n$; $l = 1/(4\varepsilon^2\delta)$; $z = 2^{-1} \bmod N$
2. $C = \emptyset$
3. $\gamma = 1 - \log \varepsilon$
4. **for** every possible assignment d_j of the γ leading bits of c **do**
5. $\quad c_{d_j} = ()$
6. $\quad w' = wg^{-2^{n-\gamma} \cdot d_j} \bmod N^2$ ("zero" the γ leading bits of c)
7. $\quad$ **for** $i = 1$ **to** $n = |N|$
8. $\quad\quad x =$**Randomize-query**$(\mathcal{O}, g, w', l)$
9. $\quad\quad$ **prepend**(x, c_{d_j})
10. $\quad\quad\quad$ **if** $x == 1$ **then**
11. $\quad\quad\quad\quad w' = w'g^{-1} \bmod N^2$
12. $\quad\quad\quad w' = w'^z \bmod N^2$
13. $\quad$ **end for**
14. $\quad$ **prepend**(d_j, c_{d_j}); $C = C \cup \{c_{d_j}\}$
15. **end for**
16. $c' \leftarrow C$
17. **return** c'.

Randomize $-$ *Query*$(\mathcal{O}, g, w, l)$

1. *countzero* $= 0$
2. *countone* $= 0$
3. **for** $i = 1$ **to** l
4. $\quad r \leftarrow Z_N$
5. $\quad s \leftarrow Z_N$
6. $\quad \hat{w} = w \cdot g^r \cdot s^N \bmod N^2$
7. $\quad x = \mathcal{O}(g, \hat{w})$
8. $\quad$ **if** $x == 0$ **then** *countzero* $++$
9. $\quad$ **else** *countone* $++$
10. **end for**
11. **if** *countzero* $>$ *countone* **then**
12. $\quad\quad$ **return** 0
13. **else**
14. $\quad\quad$ **return** 1

Fig. 2. Pseudocode description of the algorithm for the imperfect oracle case.

We go into more details. Let $\delta(n) = 1/2n$. We set $l = 1/(4\varepsilon^2(n)\delta(n))$. Notice that l is polynomial in n.

The Procedure Randomize-Query. Notice that this procedure outputs the bit by majority. Since x agrees with the correct bit with probability $1/2 + \varepsilon/2$ independently on each "experiment," we have that by the weak law of large numbers the majority counter identifies the correct bit with probability larger than $1 - \delta$. More formally, let S_Y be the number of correct answers returned by the oracle in the for loop. Similarly let S_N be the number of incorrect answers provided. By applying the weak law of large numbers we have

$$\Pr\left[\left|\frac{S_Y}{l} - \left(\frac{1}{2} + \frac{\varepsilon}{2}\right)\right| > \frac{\varepsilon}{2}\right] < \delta$$

or equivalently

$$\Pr[S_Y > S_N] > 1 - \delta.$$

The Procedure Compute-Class. We focus on the case in which we guessed the correct value d_j. Then the correct class is identified as long as each call to the procedure *Randomize-Query* returns the correct bit. This happens with probability larger than

$$(1-\delta)^n = \left(1 - \frac{1}{2n}\right)^n > \frac{1}{2}.$$

However, we do not know if we chose the correct d_j, thus the procedure outputs a random element from the set C which has cardinality 2^γ. In conclusion the correct class is output with probability $2^{-\gamma-1}$ which is still polynomial in n. □

Remark 1. Notice that the above algorithm could easily be transformed into one that runs in almost the same time but outputs c with probability nearly 1 by doing the following. Let d be an assignment of values to the top $n-\gamma$ bits of c. Once the value c_1 has been recovered using the imperfect oracle, zero it, shift c one position to the right and then re-introduce the least significant bit of d at the $(n-\gamma)$th position (note that this operation does not affect the success probability of the procedure *Randomize-Query*). Then compute c_2, perform the shift to the right and insert the second bit of d and so forth. At the very end when we have the $n-\gamma$ least significant bits of c, we just continue the loop instead of stopping it. If the guess of d was correct the additional bits will give d again. If the guess was incorrect another value for d has to be considered.

3.2. *Simultaneous Security of Many Bits*

It is not hard to show that $Class_g(\cdot)$ hides $O(\log n)$ bits simultaneously (this can be shown using standard techniques). In this section we show that by slightly strengthening the computational assumption about computing the residuosity class, we can increase the number of simultaneously secure bits, up to $O(n)$. Note that the idea of restricting the domain of a function in the context of hardness results is a well-known technique (see [11] and [14]).

What we require is that $Class_g(\cdot)$ is hard to compute even when c is chosen at random from $[0..B]$ where B is a bound smaller than N. More formally:

Definition 2. We say that computing the function $Class_g(\cdot)$ is B-hard if, for every probabilistic polynomial time algorithm $\mathcal{A}$, there exists a negligible function $\mathsf{negl}()$ such that

$$\Pr\left[\begin{array}{l} p, q \leftarrow \mathcal{PRIMES}(n/2); \quad N = pq; \\ g \leftarrow Z^*_{N^2} \text{ s.t. } ord(g) \propto N; \\ c \leftarrow [0..B]; \quad z \leftarrow Z^*_N; \quad w = g^c z^N \bmod N^2; \\ \mathcal{A}(N, g, w) = c \end{array}\right] = \mathsf{negl}(n).$$

Clearly, in order for $Class_g$ to be B-hard, it is necessary that the bound B be sufficiently large. If we had only a polynomial (in n) number of guesses, then the definition would

make no sense, since by guessing one could compute the class with probability $1/B$, which is non-negligible. Thus when we assume that $Class_g$ is B-hard we implicitly assume that $b = \log B = \omega(\log n)$.

Assumption 2. *If N is a modulus of unknown factorization, there exists no probabilistic polynomial time algorithm for the Computational Composite Residuosity Class Problem, even in the case in which we know that the output of the function is in the interval* $[0, \ldots, B]$.

Theorem 2. *Let N be a random n-bit RSA modulus; $B = 2^b$. If the function $Class_g(\cdot)$ is B-hard (see Definition 2), then it has $n - b$ simultaneously hard-core bits.*

3.3. *Proof of Theorem* 2

In order to prove Theorem 2 we first need to show that the bits in positions $1, 2, \ldots, n-b$ are individually secure. Then we prove simultaneous security.

Individual Security. Let i be an integer, $1 \leq i \leq n-b$, and assume that we are given an oracle $\mathcal{O}_i$ which on input N, g and $u \leftarrow Z^*_{N^2}$ computes correctly the ith-bit of $Class_g(u)$ with probability (over u) $1/2 + \varepsilon(n)$ where, again, $\varepsilon(n)$ is non-negligible.

In order to show that $Class_g(\cdot)$ is not B-hard, we show how to build an algorithm $\mathcal{A}$ which uses $\mathcal{O}_i$ and given $w \in Z^*_{N^2}$ with $Class_g(w) < B$, computes $c = Class_g(w)$. Let $\gamma = 1 - \log \varepsilon = O(\log n)$.

We split the proof into two parts: the first case has $1 \leq i < n - b - \gamma$. The second one is $n - b - \gamma \leq i \leq n - b$.

1. If $1 \leq i < n - b - \gamma$ the inversion algorithm works as follows. We are given $w \in Z^*_{N^2}$ where $w = g^c y^N \bmod N^2$ and we know that $c = Class_g(w) < B$. We compute c bit by bit; let c_i denote the ith bit of c. To compute c_1 we square w, i times computing $w_i = w^{2^i} \bmod N^2$. This will place c_1 in the ith position (with all zeros to its right). Since the oracle may be correct only slightly more than half of the times, we need to randomize the query. Thus we choose $r \leftarrow Z_N$ and $s \leftarrow Z^*_N$ and finally query the oracle on $\hat{w} = w_i g^r s^N \bmod N^2$. Notice the following:

- Given the assumptions on B and i we know that $w_i = w^{2^i} = g^{2^i c} z^{2^i N}$ and $2^i c$ is not taken mod N since it will not "wrap around."
- $Class_g(\hat{w}) = 2^i c + r \bmod N$. However, since $2^i c$ has at least γ leading zeros the probability (over r) that $2^i c + r$ wraps around is $\leq \varepsilon/2$.
- Since c_1 has all zeros to its right, there are no carrys in the ith position of the sum. Thus by subtracting r_i to the oracle's answer we get c_1 unless $2^i c + r$ wraps around or the oracle provides a wrong answer.

In conclusion we get the correct c_1 with probability $1/2 + \varepsilon/2$, thus by repeating several (polynomially many) times the process and taking majority we get the correct c_1 with very high probability.

Once we get c_1, we "zero" it in the squared w_i by setting $w_i \leftarrow w_i g^{-c_1 2^i} \bmod N^2$. Then we perform a "shift to the right" using Lemma 1, setting $w_i \leftarrow w_i^z \bmod N^2$ where

$z = 2^{-1} \bmod N$. At this point we have c_2 in the oracle position and we can repeat the randomized process to discover it. We iterate the above process to discover all the bits of c.[4]

Since each bit is determined with very high probability, the value $c = c_b \cdots c_1$ will be correct with non-negligible probability.

2. If $n - b - \gamma < i < n - b$ the above procedure may fail since now $2^i c$ does not have γ leading zeros anymore. We fix this problem by guessing the γ leading bits of c (i.e. $c_{b-\gamma}, \ldots, c_b$). This is only a polynomial number of guesses.

For each guess, we "zero" those bits (let α be the γ-bit integer corresponding to each guess and set $w \leftarrow w g^{-2^{b-\gamma}\alpha} \bmod N^2$). Now we are back in the situation we described above and we can run the inversion algorithm. This will give us a polynomial number of guesses for c and we output one of them randomly chosen, which will be the correct one with non-negligible probability. Notice that we are not able to verify if the solution is the correct one, but in any case the algorithm violates our security assumption (see Definition 2). A complete description of the algorithm is presented in Fig. 3.

The technical details of the proof are very similar to the previous one but we repeat them here for completeness. Again let $\delta(n) = 1/2n$ and $l = 1/(4\varepsilon^2(n)\delta(n))$. We already showed that the procedure *Randomize-Query* identifies the correct bit with probability larger than $1 - \delta$. Now we concentrate on the procedure *Compute-Class*$''$.

We distinguish two cases:

1. Case $1 \leq i < n - b - \gamma$
   In this case the correct class is computed as long as each call of the procedure *Randomize-Query* returns a correct answer. As we have shown before this happens with probability larger than $(1-\delta)^n > 1/2$. This means that with probability larger than $1/2$ the correct class is computed.
2. Case $n - b - \gamma \leq i \leq n - b$
   In this case the correct class is computed correctly (with probability larger than $1/2$) only if the correct d_j is guessed. Since there are $2^{-\gamma}$ possible choices for d_j the correct class is output with probability $2^{-\gamma-1}$.

Remark 2. Note that the same trick presented in Remark 1 can be applied here in the case when $n - b - \gamma < i < n - b$ to lower the reduction cost.

Simultaneous Security. Notice that in the above inversion algorithm, every time we query $\mathcal{O}_i$ with the value $\hat{w}$ we know all the bits in positions $1, \ldots, i-1$ of $Class_g(\hat{w})$. Indeed, these are the first $i-1$ bits of the randomizer r. Thus we can substitute the above oracle with the weaker one $\hat{\mathcal{O}}_i$ which expects $\hat{w}$ and the bits of $Class_g(\hat{w})$ in positions $1, \ldots, i-1$. □

[4] It is important to notice that Lemma 1 is necessary to perform "shifts to the right" only for the bits in positions $i = 1, \ldots, b$. For the other ones we can shift to the right by simply "undoing" the previous squaring operations. More precisely once we get c_1, instead of "zeroing" it in w_i, we can zero it in w_{i-1} (where $w_{i-1} = w^{2^{i-1}} \bmod N^2$), by setting $w_{i-1} \leftarrow w_{i-1} g^{-c_1 2^{i-1}} \bmod N^2$. In this way we take advantage of the fact that we know the intermediate squarings to perform shifts to the right.

Compute Class Algorithm (III)

ComputeClass''$(\mathcal{O}_i, w, g, N, \varepsilon, i)$

1. **if** $i > n - b$ **abort**
2. $\delta = 1/2n$; $l = 1/(4\varepsilon^2\delta)$; $z = 2^{-1} \bmod N$
3. $C = \emptyset$;
4. $\gamma = 1 - \log \varepsilon$
5. **if** $1 \leq i < n - b - \gamma$
6. $c = ()$;
7. $w' = w^{2^i} \bmod N^2$ (place c_1 in the ith position)
8. **for** $j = 1$ **to** $n = |N|$
9. $x =$ **Randomize-query**$(\mathcal{O}_i, g, w', l)$
10. **prepend**(x, c)
11. **if** $x == 1$ **then**
12. $w' = w' g^{-2^i} \bmod N^2$
13. $w = w^z \bmod N^2$
14. **end for**
15. **return** c.
16. **for** every possible assignment d_j of the γ leading bits of c **do**
17. $c_{d_j} = ()$
18. $w' = w g^{-2^{b-\gamma} \cdot d_j} \bmod N^2$ ("zero" the γ leading bits of c)
19. $w' = w'^{2^i} \bmod N^2$;
20. **for** $i = 1$ **to** $n = |N|$
21. $x =$ **Randomize-query**$(\mathcal{O}_i, g, w', l)$
22. **prepend**(x, c_{d_j})
23. **if** $x == 1$ **then**
24. $w' = w' g^{-2^i} \bmod N^2$
25. $w' = w'^z \bmod N^2$
26. **end for**
27. **prepend**(d_j, c_{d_j}); $C = C \cup \{c_{d_j}\}$
28. **end for**
29. $c' \leftarrow C$
30. **return** c'.

Fig. 3. Pseudocode description of the algorithm for the ith position case.

3.4. *Security Analysis*

We note here that the class problem can be considered a weaker version of a composite discrete log problem. Let $d = \gcd(p-1, q-1)$ and let $C_m \equiv (Z_m, +)$ denote the cyclic group of m elements, then for any t dividing λ we have

$$Z^*_{N^2} \simeq C_d \times C_{\lambda/t} \times C_{Nt}.$$

Let $g_2, g_1, g \in Z^*_{N^2}$ be the pre-images, under such an isomorphism, of generators of C_d, $C_{\lambda/t}$ and C_{Nt}, respectively. Thus we can represent any element of $Z^*_{N^2}$ uniquely as $g_2^{e_2} g_1^{e_1} g^e$, where $e_2 \in Z_d$, $e_1 \in Z_{\lambda/t}$ and $e \in Z_{Nt}$. For a given $g, g_1, g_2 \in Z^*_{N^2}$ the composite discrete logarithm problem we consider is to find these e, e_1, e_2 for any given $w \in Z^*_{N^2}$. For a given g, the class problem is to find just $e \bmod N$ for any given $w \in Z^*_{N^2}$.

Obviously if one can solve the composite discrete logarithm problem, one can solve the class problem; in particular,

$$w \equiv g_2^{e_2} g_1^{e_1} g^e \equiv g_2^{e_2} g_1^{e_1} g^{lN+x} \equiv g^x (g_2^{k_2 e_2} g_1^{k_1 e_1} g^l)^N \equiv g^x y^N \bmod N^2,$$

where $k_2 = N^{-1} \bmod d$, and $k_1 = N^{-1} \bmod \lambda/t$ (note we can make sure $x \in \{0 \cdots N\}$ by a suitable choice of l and we can force $y \in Z_N^*$ since $(y + kN)^N \equiv y^N \bmod N^2$).

However, there is a very important distinction between these two problems. In the composite discrete logarithm problem, if we are given g, g_1, g_2, e, e_1, e_2 and w we can verify (in polynomial time) that we do indeed have the discrete logarithm of x. A fascinating and open question in the class problem is to determine the complexity of an algorithm that verifies the class is correct given only g, $e \bmod N$ and w. Equation (1) shows that this is no harder than factoring, but nothing more is presently known.

Assuming that the function $Class_g$ is hard to compute even in the case that $c < B$ may seem a very strong requirement. It is in some way non-standard.

In order to justify it partially, we notice that not even a trivial exhaustive search algorithm (running in time $O(B)$) seems to work, since even if one is given a candidate c there is no way to verify that it is correct. Verification is equivalent to determining if one has an Nth residue modulo N^2, and this seems a hard problem.

Of course if one did have a verification algorithm that ran in time M, then the trivial exhaustive search method would take time $O(MB)$ and there may well be a baby-step, giant-step extension of the method that takes time $O(M\sqrt{B})$. Without an efficient verification algorithm it seems hard to exploit the fact that $c < B$.

Since this is a new assumption we are not able to make any stronger claim on its security. Further research in this direction will either validate the security assumption or lead us to learn and discover new things about the residuosity class problem. Though we note that our main theorem still holds even if there were an efficient verification algorithm (because we can choose B to be large enough to prevent $O(\sqrt{B})$ attacks).

4. A General Result

In this section we briefly show how to generalize the results from the previous section to any family of trapdoor functions with some well-defined properties. We show two theorems: the first is a direct generalization of Theorem 2; the second theorem holds for the weaker case in which we do not know the order of the group on which the trapdoor function operates. In this case we can extract fewer hard-core bits.

Let M be an m-bit odd integer, and let G be a group with respect to the operation of multiplication. Let $f\colon Z_M \to G$ be a one-way, trapdoor homomorphic function (i.e. such that $f(a + b \bmod M) = f(a) \cdot f(b) \in G$). Then we can prove that

Theorem 3. *If M is a known odd integer, under the assumption that f remains hard to invert when its input belongs to the closed interval $[0 \cdots B]$, with $B = 2^b$, f has $m - b$ simultaneously hard bits.*

It is not hard to see that the techniques of the proof of Theorem 2 can be extended to the above case.

The above theorem assumes that M is exactly known. We consider now the case in which M is not known, but we have a very close upper bound on it, i.e. we know $\hat{M} > M$ and such that $(\hat{M} - M)/M$ is negligible in m. Moreover, we need to assume that f is computable on any integer input (but taken mod M), i.e. we assume that there

is an efficient algorithm INT_f that takes as input any integer x and returns as output $INT_f(x) = f(x \bmod M)$.

Theorem 4. *Under the assumption that f remains hard to invert when its input belongs to the closed interval $[0 \cdots B]$, with $B = 2^b < \sqrt{M}$, f has $m - 2b$ simultaneously hard bits.*

Proof. The proof follows the same outline of the proof of Theorem 2 except that in this case we are not able to perform "shifts to the right" as outlined in Lemma 1 since we do not know M exactly. Thus the proof succeeds only for the bits in location $b+1, \ldots, m-b$. Notice that this implies $b < m/2$, i.e. $B < \sqrt{M}$.

Again, we first show that each bit is individually secure. We then extend this to prove simultaneous hardness.

Individual Security. Let i be an integer, $b \leq i \leq n - b$, and assume that we are given an oracle $\mathcal{O}_i$ which on input M and $u \leftarrow G$ computes correctly $(f^{-1}(u))_i$ with probability $1/2 + \varepsilon(m)$ where $\varepsilon(m)$ is non-negligible. As in the proof of Theorem 2 prove the statement by providing an algorithm $\mathcal{A}$ which uses $\mathcal{O}_i$ and given $w \in G$ with $f^{-1}(w) < B$, computes $c = f^{-1}(w)$.

The inversion algorithm works almost as the one proposed in the proof of Theorem 2. The main difference is that this time we cannot use Lemma 1 to perform shifts to the right. However, in order to let the proof go through, we adopt the following trick: once c_i is known we "zero" it in the original w by setting $w \leftarrow w \cdot INT_f(-2^{i-1}c_i)$ (where $INT_f(-2^{i-1}c_i) = f(-2^{i-1}c_i \bmod M)$, as already mentioned). We then repeat the process with the other bits. The only differences with the above process when computing c_j are that:

- We need to square w only $i - j + 1$ times (actually by saving the result of the intermediate squarings before, this is not necessary).
- To zero c_j once we found it we need to set $w \leftarrow w \cdot INT_f(-2^{j-1}c_j)$.

Since each bit is determined with very high probability, the value $c = c_b \cdots c_1$ will be correct with non-negligible probability.

The simultaneous security of the bits in positions $b, b+1, \ldots, n-b$ easily follows, as described in the proof of Theorem 2. Notice, indeed, that the same reasoning presented in the proof of Theorem 2 works here too. Consequently, the oracle $\mathcal{O}_i$ described above can be substituted with the weaker one $\hat{\mathcal{O}}_i$, that, in addition to w, requires the bits of $f^{-1}(w)$ in positions $1 \cdots i - 1$ as well.

This completes the proof. □

5. Applications to Secure Encryption

In this section we show how to construct a secure encryption scheme based on our results.

For concreteness we focus on fixed parameters, based on today's computing powers. We can assume that $n = 1024$ is the size of the RSA modulus N and $m = 128$

(the size of a block cipher key) is the size of the message M that has to be securely exchanged.

Our Solution. Using Paillier's $Class_g(\cdot)$ function with our proof methods, it is possible to hide the message M securely with a single invocation of the function. In order to encrypt 128 bits we need to set $n - b > 128$, which can be obtained for the maximum possible choice of $b = 896$ (i.e. the weakest possible assumption). In other words we need to assume that $Class_g$ is hard to invert when $c < N^{0.875}$.

To encrypt M one sets $c = r_1 \mid M$ where r_1 is a random string, chooses $y \leftarrow Z_N^*$ and sends $w = g^c y^N$. This results in two modular exponentiation for the encryption and one exponentiation to decrypt (computations are done mod N^2). The ciphertext size is $2n$.

RSA. In the case of plain RSA we can assume also that the RSA function hides only one bit per encryption (see [7]). In this scenario to encrypt (and also decrypt) the message securely we need 128 exponentiations mod N. The size of the ciphertext is $mn = 128$ Kbit. Encryption speed can be much improved by considering RSA with small public exponent. In any case our scheme is better for decryption speed and message blow-up.

Blum–Goldwasser. Blum and Goldwasser [3] show how to encrypt with the RSA/Rabin function and pay the $O(m/\log n)$ penalty only in encryption. The idea is to take a random seed r and apply the RSA/Rabin function m times to it and each time output the hard bit. Then one sends the final result r^{e^m} and the masked ciphertext $M \oplus B$ where B is the string of hard bits. It is sufficient to compute r from r^{e^m} to decrypt and this takes a single exponentiation. The size of the ciphertext is $n + m$.

A particularly efficient implementation of the Blum–Goldwasser cryptosystem is obtained, using the Rabin function [15] as the underlying trapdoor function. In order to encrypt the message M securely, 128 invocations of the function are necessary. The size of the obtained ciphertext is $n + m$. The decryption process is much faster than RSA because it requires just 768 multiplications mod p and 768 multiplications mod q plus the cost of combining the two results using the Chinese Remainder Theorem. Note that even if the total number of multiplications is comparable with the number of multiplications required to decrypt in our solution, the Blum–Goldwasser scheme is still more efficient due to the fact that all the multiplications are performed modulo integers whose size is approximately one-quarter of the size of the modulus used for the Paillier function.

We clearly lose compared with this scheme.

Remark 3. It is worthwhile noticing that even if the proposed solution is less efficient in practice than the Blum–Goldwasser one, it remains asymptotically better. As a matter of fact, we need only $O(m/k)$ (where $k = \omega(\log n)$ is the number of simultaneously hard bits produced) invocations of the trapdoor function, while all previously proposed schemes require many more invocations (in general, the number of invocations, has order $O(m/\log n)$). Basically for longer messages we may "catch up" with the other schemes.

The observed slow down, depends solely on the fact that the function used is less efficient than RSA or Rabin. It would be nice to come up with more efficient trapdoor functions that also hide many bits.

6. Conclusions

In this paper we presented the bit security analysis of the encryption scheme proposed by Paillier at Eurocrypt '99 [13]. We prove that the scheme hides the least significant bit of the N-residuosity class. Also by slightly strengthening the computational assumption about residuosity classes we can show that Paillier's encryption scheme hides up to $O(n)$ bits.

An interesting theoretical implication of our results is that we presented the first candidate trapdoor functions that hide many (up to $O(n)$) bits. No such object was known previously in the literature.

There are several problems left open by this research. Are there trapdoor functions that hide $\omega(\log n)$ bits and are comparable in efficiency to RSA/Rabin? In the case of RSA/Rabin can we come up with a "restricted input assumption" that will allow us to prove that they also hide $\omega(\log n)$ bits? Regarding our new assumptions: is it possible to devise an algorithm to compute $Class_g(\cdot) < B$ that depends on B?

References

[1] W. Alexi, B. Chor, O. Goldreich and C. Schnorr. RSA and Rabin Functions: Certain Parts Are as Hard as the Whole. *SIAM J. Comput*., 17(2):194–209, April 1988.

[2] J.C. Benaloh. Verifiable Secret-Ballot Elections. Ph.D. Thesis, Yale University, 1988.

[3] M. Blum and S. Goldwasser. An Efficient Probabilistic Public-Key Encryption Scheme Which Hides All Partial Information. In *Proc. Crypto* '84, LNCS vol. 196, pages 289–302.

[4] M. Blum and S. Micali. How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SIAM J. Comput*., 13(4):850–864, 1984.

[5] D. Catalano, R. Gennaro and N. Howgrave-Graham. The Bit Security of Paillier's Encryption Scheme and Its Applications. In *Advances in Cryptology - Eurocrypt* '01, LNCS vol. 2045, pages 229–243. Springer-Verlag, Berlin, 2001.

[6] D. Catalano, R. Gennaro, N. Howgrave-Graham and P.Q. Nguyen. Paillier's Cryptosystem Revisited. In *Proc. 8th ACM Conference on Computer and Communication Security*, pp. 206–214, 2001.

[7] R. Fischlin and C.P. Schnorr. Stronger Security Proofs for RSA and Rabin Bits. *J. Cryptology*, 13(2):221–244, Spring 2000.

[8] O. Goldreich and L. Levin. A Hard-Core Predicate for All One-Way Functions. In *Proc. 21st ACM Symposium on Theory of Computing*, 1989.

[9] S. Goldwasser and M. Bellare. Lecture Notes in Cryptography. Available on-line at `www-cse.ucsd.edu/mihir/crypto-lecnotes.html`.

[10] S. Goldwasser and S. Micali. Probabilistic Encryption. *J. Comput. System Sci*., 28(2):270–299, April 1984.

[11] J. Hastad, A.W. Schrift and A. Shamir. The Discrete Logarithm Modulo a Composite Hides $O(n)$ Bits. *J. Comput. System Sci*., 47:376–404, 1993.

[12] T. Okamoto and S. Uchiyama. A New Public-Key Cryptosystem as Secure as Factoring. In *Advances in Cryptology - Eurocrypt* '97, LNCS vol. 1233, pages 308–318. Springer-Verlag, Berlin, 1997.

[13] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology - Eurocrypt* '99, LNCS vol. 1592, pages 223–238. Springer-Verlag, Berlin, 1997.

[14] S. Patel and G.S. Sundaram. An Efficient Discrete Log Pseudo Random Generator. In *Advances in Cryptology - CRYPTO '*98, LNCS vol. 1492, pages 304–315. Springer-Verlag, Berlin, 1998.
[15] M. Rabin. Digital Signatures and Public Key Encryptions as Intractable as Factorization. MIT Technical Report No. 212, 1979
[16] R. Rivest, A. Shamir and L. Adelman. A Method for Obtaining Digital Signature and Public Key Cryptosystems. *Comm. ACM*, 21:120–126, 1978.
[17] A. Yao. Theory and Applications of Trapdoor Functions. In *Proc. FOCS* '82, pages 80–91, 1982.