The Bit Security of Paillier's Encryption Scheme and its Applications *

Dario Catalano[1], Rosario Gennaro[2] and Nick Howgrave-Graham[2]

[1] Dipartimento di Matematica e Informatica
Università di Catania. Viale A. Doria 6, 95125 Catania.
Email: catalano@dmi.unict.it.
[2] IBM T.J.Watson Research Center
PO Box 704, Yorktown Heights, New York 10598, USA.
Email: {rosario,nahg}@watson.ibm.com

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 hard 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.

1 Introduction

At EuroCrypt'99 Paillier [10] proposed a new encryption scheme based on higher residuosity classes. It generalized previous work by Okamoto and Uchiyama [9]. Both works are based on the problem of computing high-degree residuosity classes modulo a composite of a special form (in Paillier's the modulus is N^2 where N is a typical RSA modulus, while in [9] the modulus is $N = p^2q$ where p, q are large primes.)

* The first author's research was carried out while visiting the Computer Science Department of Columbia University.

The mathematical details are described below, but for now let us 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 c^{th} 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 it 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 [7]) 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 $f : \{0,1\}^n \to \{0,1\}^n$ we say that $\pi : \{0,1\}^n \to \{0,1\}$ 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 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 in [1]. Goldreich and Levin in [6] 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 the $c = Class(w)$ 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 remain 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 function 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 in [7]).

However, for all previously known trapdoor functions (like RSA) $f : \{0,1\}^n \rightarrow \{0,1\}^n$ 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 hide up to $O(n)$ bits. To our knowledge this is the first example of trapdoor problems with a super-logarithmic number of hard-core predicates.

We also generalize our construction 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 [10] 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 his scheme hide *all* n input bits.

Notice however the difference with our result. We prove that these two schemes hide many bits, under a *computational* assumption, about 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 (only) require that the adversary cannot compute the *full* solution of a computational problem. Thus, whenever possible, computational assumptions should be preferred to decisional ones.

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

[1] The above discussion implicitly rules out *iterated* functions. Indeed [4] shows 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 hide the following k bits: $\pi(x), \pi(f(x)), ..., \pi(f^{k-1}(x))$. We are interested in functions that hide several bits in a *single* iteration.

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 $c < N^{.875}$.

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 [7] 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 $\phi(N)$. 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-Uchiyama in [9] and Paillier in [10] were able to lift this restriction and consider higher degree residuosity classes.

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.* [8]. 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). [8] shows the first example of one-way function with a superlogarithmic number of hard-core bits. No such examples was known for *trapdoor* function.

Building on ideas from [8], Patel and Sundaram in [11] show that if one assumes that $f(x) = g^x \bmod p$ (with p prime) remains hard to invert even when $x < B$, then discrete logarithm simultaneously hide $k - b$ 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 [11] and our computational assumption though. In [11] 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 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 to 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 $\mathsf{negl} : \mathbf{N} \rightarrow \mathbf{R}^+$ is *negligible* iff for every polynomial $P(n)$ there exists a $n_0 \in \mathbf{N}$ 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.

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. Let $f_n : \{0,1\}^n \rightarrow \{0,1\}^n$ be a family of permutations. We say that f_n is a *trapdoor family* if the following conditions hold:

- f_n can be computed in polynomial time (in n)
- f_n can be inverted in polynomial time only if given a description of f_n^{-1}. I.e. for any probabilistic polynomial time Turing Machine $\mathcal{A}$ we have that

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

The above notion can be generalized to *probabilistic* functions where each $f_n : \{0,1\}^n \times \{0,1\}^r \rightarrow \{0,1\}^{n+r}$ 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 $y = f(x)$ guesses $\pi(x)$ with probability substantially better than $1/2$. More formally for any probabilistic polynomial time Turing Machine $\mathcal{A}$ we have that

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

For one-way functions f_n, a possible way to prove that a predicate π is hard is to show that any 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 $\mathcal{A}'$ 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 [14]) is to say that it is not possible to guess the value of the j^{th} 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:

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

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 [10]. Readers are referred to [10] for details and proofs which are not given here.

Preliminaries. Let $N = pq$ be an RSA modulus, i.e. 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. Let us denote with $\mathcal{B}$ the set of such elements. It can be shown that g induces a bijection

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

Thus, given g, for an element $w \in Z^*_{N^2}$ there exists an 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

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

It can be shown that if the factorization of N is known then one could solve this problem: indeed let $\lambda = \mathrm{lcm}(p-1, q-1)$ 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[2] $L(u) = (u-1)/N$.

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 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 [6] 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. We show that, 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.

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

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

for some $y' \in Z^*_N$

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

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

Observe that, being the group Z_{N^2} isomorphic to $Z^*_N \times Z_N$ (for $g \in \mathcal{B}$) [10], this is enough to conclude the proof. □

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 to have access to a

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

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 will suppose $Pr_w[\mathcal{O}(g,w) = lsb(c)] \geq \frac{1}{2} + \epsilon(n)$ where $\epsilon(n) > \frac{1}{p(n)}$, for some polynomial $p(\cdot)$. For convenience we will denote $\epsilon(n)$ by simply ϵ in the following analysis.

The Perfect Case. The algorithm computes c, bit by bit starting from $lsb(c)$. Denote $c = c_n \ldots c_2 c_1$ 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 homorphic 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 follows(where () denotes the empty string and $\alpha|\beta$ is the concatenation of the bit strings α and β):

$ComputeClass(\mathcal{O}, w, g, N)$
1. $z = 2^{-1} \bmod N$
2. $c = ()$
3. **for** $i = 0$ **to** $n = |N|$
4. $\quad x = \mathcal{O}(g,w)$
5. $\quad c = c|x$
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

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 choosing randomly $r \in_R Z_N$ and $s \in_R Z_N^*$, considering $\hat{w} = w \cdot g^r \cdot s^N$ and querying $\mathcal{O}(g, \hat{w})$ on several randomized $\hat{w}$'s.

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 $\gamma = 1 - \log \epsilon$ 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 $2^{-\gamma} = \epsilon/2$. We can add this probability to the error probability of the oracle. Consequently the oracle is now correct with probability $\frac{1}{2} + \frac{\epsilon}{2}$. 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 obtain a (polynomially sized) set of candidate values for c. Notice that we cannot check, given w, which one of the c's is the correct one. However this still implies an algorithm to output c correctly with probability $1/poly$, which contradicts Definition 1. □

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 strenghtening the computational assumption about computing residuosity class then we can increase the number of simultaneously secure bits, up to $O(n)$.

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

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

Clearly in order for the $Class_g$ 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 be clearly false. Thus when we assume that $Class_g$ is B-hard we implicitly assume that $b = \log B = \omega(\log n)$.

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 $u \in_R Z^*_{N^2}$ computes correctly the i^{th}-bit of $Class_g(u)$ with a probability (over u) of $1/2 + \epsilon(n)$, where again $\epsilon(n) > 1/poly(n)$.

In order to show that $Class_g(\cdot)$ is not B-hard, we will 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 $\gamma = 1 - \log \epsilon = O(\log n)$.

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

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 i-th 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 i-th position (with all zeroes 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 $r \in_R Z_N$ and $s \in_R Z^*_N$ 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".
- $Class_g(\hat{w}) = 2^i c + r \bmod N$. But since $2^i c$ has at least γ leading zeroes the probability (over r) that $2^i c + r$ wraps around is $\leq \epsilon/2$.
- Since c_1 has all zeroes to its right, there are no carrys in the i-th 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 $1/2 + \epsilon/2$, 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 [3].

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

If $n - b - \gamma < i < n - b$ the above procedure may fail since now $2^i c$ does not have γ leading zeroes 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.)

[3] We note that Lemma 1 is necessary to perform "shifts to the right" only for the bits in position $i = 1, \ldots, b$. For the other ones we can shift to the right by simply "undoing" the previous squaring operations.

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 position $1, \ldots, i-1$ 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 position $1, \ldots, i-1$. □

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 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 preimages, 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

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

where $k_2 = N^{-1} \bmod d$, and $k_1 = N^{-1} \bmod \lambda/t$, where we note we can make sure $x \in \{0 \ldots N\}$ 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 the class problem and the discrete log problem. 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 partially justify it, 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 N'th 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 took time $O(M\sqrt{B})$.

Without an efficient verification algorithm it seems hard to exploit the fact that $c < B$.

Of course because 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 (because we can choose B to be large enough to prevent $O(\sqrt{B})$ attacks) even if there were an efficient verification algorithm.

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 are able to extract less hard-core bits.

Theorem 3. *Let M be an m-bit odd integer, and G a group with respect to the operation of multiplication. Let $f : Z_M \to G$ be a one-way, trapdoor isomorphic function (i.e. such that $f(a + b \bmod M) = f(a) \cdot f(b) \in G$). Then, under the assumption that f remains hard to invert when its input belongs to the closed interval $[0 \dots B]$, with $B = 2^b$, it follows that 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. Let us now consider the case in which M is not known, but we have a very close upper bound on it. I.e. we know $\hat{M} > M$ such that $\frac{\hat{M}-M}{M} = \mathsf{negl}(m)$. Moreover we assume that f is computable on any integer input (but taken mod$M$), i.e. we assume that there is an efficient algorithm A that takes as input any integer x and returns as output $A(x) = f(x \bmod M)$.

Theorem 4. *Under the assumption that f remains hard to invert when its input belongs to the closed interval $[0 \dots B]$, 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 $b + 1, \dots, m - b$ 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 $u \in_R G$ computes correctly

$(f^{-1}(u))_i$ with probability $1/2 + \epsilon(m)$ where $\epsilon(m) > 1/poly(m)$. 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 $w \leftarrow wf(-2^{i-1}c_i \bmod M)$. 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 $w \leftarrow wf(-2^{j-1}c_j \bmod M)$

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

The simultaneous security of the bits in position $b, b+1, \dots, n-b$ easily follows, as described in the proof of theorem 2. Details will appear in the final version of this paper.

5 Applications to Secure Encryption

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

For concreteness let us 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) to be 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 securely hide the message M 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 $c < N^{.875}$.

To encrypt M one sets $c = r_1|M$ where r_1 is a random string, chooses $y \in_R Z_N^*$ 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 [5]). In this scenario to securely encrypt (and also decrypt) the message 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 in [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 $n + m$.

Using the Rabin function this costs only 128 multiplications to encrypt and a single exponentiation to decrypt. We clearly lose compared to this scheme.

Remark: It is worth to notice 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, solely depends 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 hides many bits.

6 Conclusions

In this paper we presented the bit security analysis of the encryption scheme proposed by Paillier at Eurocrypt'99 [10]. We prove that the scheme hides the least significant bit of the N-residuosity class. Also by slightly strenghtening 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. Computing, 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. *Proc. of 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 Journal on Computing*, Vol. 13, No. 4:850-864, 1984
5. R. Fischlin and C.P. Schnorr. Stronger Security Proofs for RSA and Rabin Bits. *J. of Cryptology*, 13(2):221–244, Spring 2000.
6. O. Goldreich and L. Levin A hard-core predicate for all one-way functions. *Proc. 21^{st} ACM Symposium on Theory of Computing*, 1989
7. S. Goldwasser and S. Micali. Probabilistic Encryption. *JCSS*, 28(2):270–299, April 1984.
8. J. Hastad, A. W. Schrift and A. Shamir. The Discrete Logarithm Modulo a Composite Hides $O(n)$ Bits. *JCSS* Vol. 47, pages 376-404, 1993.
9. T. Okamoto and S. Uchiyama. A New Public-Key Cryptosystem as Secure as Factoring In *Advances in Cryptology - Eurocrypt '97*, LNCS vol. 1233, Springer, 1997, pages 308-318.
10. P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology - Eurocrypt '99*, LNCS vol. 1592, Springer, 1997, pages 223-238.
11. S. Patel and G. S. Sundaram. An Efficient Discrete Log Pseudo Random Generator. In *Advances in Cryptology - CRYPTO '98*, LNCS vol. 1492, Springer, 1998, pages 304-315.
12. M. Rabin. Digital Signatures and Public Key Encryptions as Intractable as Factorization. MIT Technical Report no. 212, 1979
13. R. Rivest, A. Shamir and L. Adelman. A Method for Obtaining Digital Signature and Public Key Cryptosystems. *Comm. of ACM*, 21 (1978), pp. 120–126
14. A. Yao. *Theory and Applications of Trapdoor Functions.* IEEE FOCS, 1982.