Evaluation Report on the **HIME(R) Cryptosystem**

Jacques Stern

1 Introduction

This document is an evaluation of the **HIME(R) Cryptosystem**. Our work is based on the analysis of documents [17, 18], and on various documents from the literature. The present report is organized as follows: firstly, we briefly review the cryptosystem; next we discuss the security level of the cryptographic primitive which underlies the scheme and analyze its relation to the difficulty of factoring; then, we evaluate the security level of the scheme itself in the light of strong security notions such as semantic security and security against adaptive chosen-ciphertext attacks; and finally, we compare HIME(R) with other schemes based on factoring. This is as requested by IPA.

2 Brief Description of the Scheme

2.1 Specification Review

HIME(R) is based on the hardness of factoring integers N of the form $p^d q$, where p and q are prime numbers — of the same size — such that $p = 3 \bmod 4$, $q = 3 \bmod 4$, and d is a small integer > 1 (typically $d = 2$). Let k be the size in bits of integer N, $2^{k-1} < N < 2^k$. The basic function f on which HIME(R) relies is defined by

$$\begin{array}{ccc} f : \{0,1\}^{k-1} & \longrightarrow & \mathbb{Z}_N \\ x & \longmapsto & x^2 \bmod n \end{array}$$

We still denote by f the extension of this function to $\mathbb{Z}_N$ and state the following:

Theorem 1 *Let y be a quadratic residue in $\mathbb{Z}_N^\star$, where $N = p^d q$. There are exactly four integers x_1, x_2, x_3, x_4, $0 < x_1 < x_2 < N/2 < x_3 < x_4 < N$, such that $f(x_i) = y \bmod N$.*

Proof. The result is well known. However, we include the proof in order to provide the actual computation of the four square roots, shown in [17, page 10].

Square roots modulo N are obtained by combining the two square roots modulo q and the two square roots modulo p^d, by means of the chinese remainder theorem (CRT). Now, it is easy to check that, when $p = 3 \bmod 4$, the square roots of y modulo q are $\pm y^{(q+1)/4} \bmod q$. Similarly, the square roots of y modulo p are $\pm y^{(p+1)/4} \bmod p$. Thus, it remains to see how one can lift the latter modulo p^d.

Write any square root x of y modulo p^d in base p as:

$$x = x_0 + x_1 p + \ldots + x_{d-1} p^{d-1}, \text{ with } 0 \leq x_i < p.$$

We show how to recursively compute x_i from $x_0, \cdots x_{i-1}$. For any i, $1 \leq i \leq d-1$, we let $X_i = x_1 + \ldots + x_{i-1} p^{i-2}$, and compute modulo p^{i+1}:

$$y = x^2 = (x_0 + X_i p + x_i p^i)^2 = (x_0 + X_i p)^2 + 2 x_0 x_i p^i \bmod p^{i+1}.$$

This uniquely defines x_i as:

$$x_i = \frac{y - (x_0 + X_i p)^2 \bmod p^{i+1}}{p^i} \times (2x_0)^{-1} \bmod p.$$

Conversely, it is easy to check that, using the above formula for x_i, one can obtain a square root modulo p^d from each of the two square roots modulo p.

To conclude, it is enough to observe that the four square roots modulo $p^d q$ are pairwise opposite. Thus two of them, say $x_1 < x_2$, are $< N/2$ and the other two, $x_3 < x_4$, are $> N/2$. □

Before explaining how the HIME(R) cryptosystem uses the above function, we introduce a more formal framework, that will be useful when we later perform the security analysis.

2.1.1 Public-Key Encryption Schemes

A public-key encryption scheme on a message space $\mathcal{M}$ consists of three algorithms $(\mathcal{K}, \mathcal{E}, \mathcal{D})$:

- the *key generation algorithm* $\mathcal{K}(1^k)$, which receives as its input a security parameter k and outputs, in a probabilistic manner, a pair of matching private-public keys $(\mathsf{sk}, \mathsf{pk})$
- the *encryption algorithm* $\mathcal{E}_{\mathsf{pk}}(m; r)$, which outputs a ciphertext c corresponding to the plaintext $m \in \mathcal{M}$, using random coins $r \in \Omega$
- the *decryption algorithm* $\mathcal{D}_{\mathsf{sk}}(c)$, which outputs the plaintext m associated to the ciphertext c, or a Reject answer if the ciphertext is invalid.

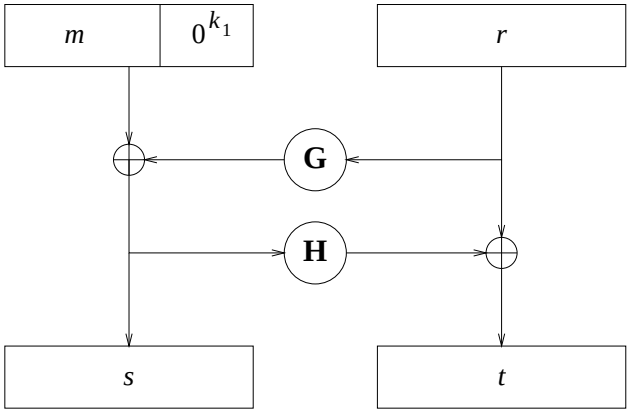


Figure 1: OAEP padding

2.1.2 The HIME(R) Cryptosystem

The key generation algorithm $\mathcal{K}(1^k)$ of the HIME(R) cryptosystem produces a k-bit integer N as a product $N = p^d q$, where d is a small integer strictly greater than 1, and p and q are odd primes of same length, such that $p = 3 \bmod 4$, and $q = 3 \bmod 4$.

It chooses parameters k_0, k_1 and n, such that $n = k - k_0 - k_1 - 1$, and $2k_0 < k$. It also chooses two hash functions G and H:

$$G : \{0,1\}^{k_0} \longrightarrow \{0,1\}^{k-k_0-1} \text{ and } H : \{0,1\}^{k-k_0-1} \longrightarrow \{0,1\}^{k_0}.$$

The public key pk is the tuple $(N, k, k_0, k_1, \mathsf{G}, \mathsf{H})$. The private key sk is the factorization (p, q) of N, which helps inverting function f, as shown by theorem 1. The message space is $\mathcal{M} = \{0,1\}^n$, and, finally, the space of random coins for the encryption algorithm is $\Omega = \{0,1\}^{k_0}$.

The encryption algorithm $\mathcal{E}_{\mathsf{pk}}(m; r)$ takes $m \in \mathcal{M}$ and a random value $r \stackrel{R}{\leftarrow} \Omega$, and computes a $k-1$-bit integer $x = s \,\|\, t$, by OAEP formatting (see [5] and Figure 1), as follows:

$$s = (m \,\|\, 0^{k_1}) \oplus \mathsf{G}(r) \text{ and } t = r \oplus \mathsf{H}(s)$$

where $\oplus$ denotes bitwise addition modulo 2, and $\|$ denotes concatenation of bit strings. The output ciphertext is $c = f(x)$, with $x = s \,\|\, t \in \{0,1\}^{k-1}$.

The decryption algorithm $\mathcal{D}_{\mathsf{sk}}(c)$ extracts the 4 square roots of c, denoted x_i (for $i = 1, \ldots, 4$), and parses them as $x_i = s_i \,\|\, t_i$, with $s_i \in \{0,1\}^{n+k_1}$ and $t_i \in \{0,1\}^{k_0}$, if $x_i \in \{0,1\}^{k-1}$. Then, it computes:

$$r_i = t_i \oplus \mathsf{H}(s_i) \text{ and } M_i = s_i \oplus \mathsf{G}(r_i).$$

For each index i, it checks whether M_i is correctly formatted, with its k_1 trailing bits

zero. As soon as it finds such a properly formatted M_i, it returns the n leading bits of M_i. If no properly formatted M_i is found, the decryption algorithm returns Reject.

We refer to [17, pages 13–14] for a precise definition of G and H and for references on the exact choice of the parameters.

3 Security Level of the Cryptographic Primitive

In this section, we investigate the security of the underlying cryptographic primitive, both in terms of complexity-theoretic reductions and with respect to the recommended parameters.

3.1 Complexity-Theoretic Arguments

We show that function f is *one-way*, based on the hardness of factoring integers N of the form $N = p^d q$.

Theorem 2 *Let N be an integer $N = p^d q$, where p and q are primes of the same size k. Let f be the associated squaring function. If an adversary $\mathcal{A}$ is able to invert f with probability ε, within time bound t, then, there is a machine $\mathcal{B}$ that can factor N with probability $\varepsilon' \geq \varepsilon/3$, within time bound $t' \leq t + T_{\mathsf{gcd}}(k)$, where $T_{\mathsf{gcd}}(k)$ denotes the time for computing a* gcd *between k-bit integers:*

$$\mathsf{Succ}^{\mathsf{ow}}_f(t) \leq 3 \times \mathsf{Succ}^{\mathsf{fact}}(k, t + T_{\mathsf{gcd}}(k)).$$

In the above, $\mathsf{Succ}^{\mathsf{ow}}_f(t)$ denotes the maximal success probability of inverting f, for any adversary whose running time is bounded by t. Similarly, $\mathsf{Succ}^{\mathsf{fact}}(k, t)$ denotes the maximal success probability of factoring a k-bit integer $N = p^d q$, for any adversary whose running time is bounded by t.

Proof. Let us consider an adversary $\mathcal{A}$ able to break the one-wayness of f with probability ε, within time bound t, in symbols:

$$\mathsf{Succ}^{\mathsf{ow}}_f(\mathcal{A}) \stackrel{\mathsf{def}}{=} \Pr[x \stackrel{R}{\leftarrow} \{0,1\}^{k-1}, y \leftarrow f(x), z \leftarrow \mathcal{A}(y) : z \in \{0,1\}^{k-1} \wedge f(z) = y] \geq \varepsilon$$

where probabilities include the internal random tape of the probabilistic machine $\mathcal{A}$.

We use the above adversary $\mathcal{A}$ to factor N, by describing a machine $\mathcal{B}$ that makes calls to $\mathcal{A}$:

1. $\mathcal{B}$ chooses $x \stackrel{R}{\leftarrow} \{0,1\}^{k-1}$;
2. $\mathcal{B}$ computes $y \leftarrow f(x)$;
3. $\mathcal{B}$ runs $\mathcal{A}$ on y, and gets z;

4. $\mathcal{B}$ returns $\gcd(N, x+z)$.

We expect that the output value is either p^d or q. More accurately, we need to obtain a lower bound for the probability ν that $\mathcal{B}$ returns a non trivial factor of N:

$$\nu = \Pr[x \stackrel{R}{\leftarrow} \{0,1\}^{k-1}, y \leftarrow f(x), z \leftarrow \mathcal{A}(y) : f(z) = y \wedge z \neq \pm x \bmod N].$$

Recall from theorem 1 that any quadratic residue y has four square roots, x_1, x_2, x_3, x_4, such that

$$0 < x_1 < x_2 < N/2 < x_3 < x_4 < N,$$

and $x_1 = N - x_4$, $x_2 = N - x_3$. Note that x_1, x_2 are $< 2^{k-1}$. Thus, there are $j = 2$, 3 or 4 square roots $< 2^{k-1}$, which means that the input x to $\mathcal{B}$ takes $j = 2$, 3 or 4 possible values. On the other hand, $\mathcal{B}$ is successful is $x \neq \pm z$, where z is the output. Inspecting all cases, we see that this means one value for x when $j = 2$, one or two values for x when $j = 3$, and at least two values for x, when $j = 4$. Altogether, the worst case is one out of three, and therefore:

$$\nu \geq \frac{1}{3} \times \mathsf{Succ}_f^{\mathsf{ow}}(\mathcal{A}) \geq \frac{\varepsilon}{3}.$$

This completes the proof of the theorem. □

3.2 Size of the Parameters

As was just observed, the security of the basic scheme, consisting in squaring a given value, is essentially equivalent to the hardness of factoring integers N of the form $N = p^d q$, even if there is a small security loss in terms of exact security.

Thus, the security of the basic scheme is basically equivalent to the hardness of factoring integers N of the form $N = p^d q$. It is unclear whether or not factoring is easier for such numbers than it is in case of integers with two prime factors. In order to state an opinion, we briefly review the performances of known factoring algorithms. Such algorithms fall into three families, according to their sensitivity to the size of the factors and to the existence of repeated factor. Such a review is also provided in [18] in order to make some recommendations about the size of the parameters.

Before entering into a more precise discussion, let us mention that the idea of having moduli of the form $p^d q$ is not new. For example, it appears in [27]. This remark is not in terms of intellectual property but rather in terms of the novelty of the idea.

3.2.1 Factoring Techniques Sensitive to the Size of the Smaller Factor

Pollard's ρ-Method. The idea behind the method is to iterate a polynomial P with integer coefficients that is to say computing $x_1 = P(x_0)$, $x_2 = P(P(x_0))$, etc. In time

complexity $\mathcal{O}(\sqrt{p})$, where p is the smallest prime factor of N, one finds a collision modulo p, i.e. two values x_i and x_j, $i \neq j$, such that $x_i = x_j \bmod p$. Computing $\gcd(x_i - x_j, N)$ factors N.

Although there are several optimizations, the ρ-method can only be used to cast out "small" factors of an integer (say 30-digit factors). As far as we know, it has not been used to find significantly larger factors.

The $p-1$ Method. Let B be a positive integer. A number is B-smooth if it is of a product of prime numbers all $< B$. B-smooth numbers are usually used through a table of primes $< B$. The $p-1$ method relies on the use of Fermat's little theorem: if $p-1$ is B-smooth, then the computing $\gcd(N, a^{\ell(B)} - 1)$ factors N, where $\ell(B)$ is the product of all prime factors $< B$.

The security against this factoring method is adequately addressed by the requirement that each of $p-1$, $q-1$ has a large prime factor in [17, page 14].

The Elliptic Curve Method. The ECM is a generalization of the $p-1$ method, for which the above simple countermeasure is not sufficient. Consider an elliptic curve mod$N$ with equation

$$y^2 = x^3 + ax + 1.$$

If the number of points of this curve modulo p is B-smooth, then a factor of N can be discovered along the computation of the scalar multiplication of $M_0 = (0, 1)$ by $\ell(B)$, according to the group law of the elliptic curve.

The success probability of the algorithm is as follows: Let

$$L(x) = \exp(\sqrt{\ln x \ln \ln(x)}),$$

then, the curve is $L(p)^{\alpha}$-smooth with probability $L(p)^{-1/(2\alpha)+o(1)}$. This is minimal for $\alpha = 1/\sqrt{2}$ and gives an expected running time of $L(p)^{\sqrt{2}+o(1)}$ group operations on the curve.

There have been several improvements of ECM factoring, notably the FFT extension of P. Montgomery. Furthermore, several implementations of ECM are available. The current ECM factoring record was established in December 1999, when a prime factor with 54 digits of a 127-digit composite number N was found with GMP-ECM, a free implementation of the Elliptic Curve Method (see [20]). The limit used was $B = 15,000,000$.

In a recent paper [8], Richard Brent extrapolates the ECM record to be of D digits at year about

$$Y = 9.3 * \sqrt{D} + 1932.3$$

this would give records of $D = 60$ digits at year $Y = 2004$ and $D = 70$ at year 2010. Such record would need $B \simeq 2,900,000,000$ and require testing something like $340,000$

curves. It can be noted that, if Brent's prediction is correct, parameters of the scheme under review will become insecure at year $Y = 2045$ (primes over 488 bits, and thus $D = 146$, see [17, page 13]).

3.2.2 Factoring Techniques which are not Sensitive to the Size of the Smaller Factor

Quadratic Sieve. The quadratic sieve method (QS) factors N by gathering many congruences of the form

$$x^2 = (-1)^{e_0} p_1^{e_1} \cdots p_m^{e_m}$$

where $p_1, \cdots, p_m$ is a list of prime numbers $< B$, called the factor base. This is done by finding B-smooth numbers of the form $Q(a) = (\sqrt{N}+a)^2 - N$. It turns out that there is a very efficient sieving process that performs the job without division, hence the name QS. Once enough congruences have been gathered, one obtains another congruence of the same type with all exponents e_i even: this is done by gaussian elimination mod2. Thus one gets some relation $x^2 = y^2 \bmod N$ and, with significant probability, computing $\gcd(x - y, N)$ factors N. The time complexity of QS is $\mathcal{O}(L(N)^{1+o(1)})$ but, as it uses very simple operations, it is usually more efficient than ECM for numbers whose smallest prime factor exceeds $N^{1/3}$.

Many improvements of the basic method have been found, notably the multiple polynomial variation (MPQS) and the large prime variation. This has led to very efficient implementation and, until the mid-nineties, was used to set up factoring records. The largest number factored by MPQS is the 129-digit number from the "RSA Challenge" (see [25]). It was factored in April 1994 and took approximately 5000 MIPS-years (see [1]).

Number Field Sieve. The number field sieve (NFS) is somehow similar to the QS but it searches for congruences in some number field (algebraic extension of the rational numbers). The method uses two polynomials with a common root m modulo N. These polynomials should have as many smooth values as possible. The time complexity of NFS is

$$\mathcal{O}(e^{(\ln N)^{1/3}(\ln \ln N)^{2/3}(C+o(1))})$$

for a small constant C (about $(64/9)^{1/3} \simeq 1.923$). This is asymptotically considerably better than QS. In practical terms, NFS beats QS for numbers of more than about 110 digits (see [14]). The number field sieve was used to factor the 130-digit RSA challenge number in April 1996, with an amount of computer time which was only a fraction of what was spent on the old 129-digit QS-record. It was later used to factor RSA-140 in February 1999 with an amount of computer time about 2000 MIPS-years [9]. In august 1999, the factorization of RSA-155 from the RSA list was obtained [10]. The amount of computer time spent on this new factoring world record is equivalent to 8000

MIPS-years, whereas extrapolation based on RSA-140 and the asymptotic complexity formula for NFS predicted approximately 14000 MIPS-years. The gain was caused by an improved polynomial search method. The final linear algebra part took 224 CPU hours and 2 Gbytes of central memory on a Cray C916.

The current record is still in the 512 bit range. However, factoring technology has spread out: another 512 bit number has been factored in October 2000, by a swedish team [2]. The experiment was carried through by a group which did not consist of experts in the area, and, despite the fact that the running time that they used was not optimal, they were successful. The current factoring record [3] is the completion of the factorization of $2^{953}+1$ by factoring the remaining divisor called c158. This was achieved in january 2002. Thus, it took more than two years to gain 3 decimal digits.

The main obstacle to a fully scalable implementation of NFS is actually the linear algebra, although progress has been made (see [21]). In [10], the authors derive the following formula

$$Y = 13.24D^{1/3} + 1928.6$$

for predicting the calendar year for factoring D-digit number by NFS. The same formula appears in [8] and produces $Y = 2026$ for $D = 404$, i.e. for a 1344 bit modulus, as proposed in HIME(R) [17, page 13].

3.2.3 Specific Factoring Techniques for Numbers of the Form $p^d q$

In recent work (see [6]), a new factoring method that applies to integers of the form $p^d q$ has been found. The method is based on an earlier result of Coppersmith (see[12]), showing that an RSA modulus $N = pq$, with p, q of the same size, can be factored given half the most significant bits of p. It turns out that, for numbers of the form $N = p^d q$, with p, q of the same size, fewer bits are needed.

Note that disclosing the leading bits of p provides a rough approximation P of p. What remains to be found is the difference $x = p - P$. The new method is based on finding polynomials with short enough integer coefficients, which vanish at x modulo some power p^{dm} of p. Such polynomials are actually zero at x. Thus, factoring is achieved by finding the appropriate root. The polynomial itself is computed by the LLL lattice reduction algorithm from [19]. LLL is run on lattices of dimension d^2 with basis vectors of size $\mathcal{O}(d \log N)$. Let γ be the corresponding computing time. Taking into account the work-factor tied with guessing the approximation of p, the total running time is

$$2^{\frac{c+1}{d+c}.\log p}.\gamma$$

where c is such that $q \simeq p^c$.

Comparing the above estimate with the running time for ECM, one can see that the new method beats ECM for d larger than, approximately $\sqrt{\log p}$. For the suggested parameters of HIME(R), which set $d = 2$, the algorithm is certainly impractical.

3.2.4 Conclusion

Based on current estimates, it appears that the proposed parameters for HIME(R) should remain secure for at least twenty years.

4 Security Analysis

In this section, we briefly recall the formal framework of security proofs for public key cryptosystem. Next, we review the security proof from [18], and point out problems in this proof. Finally, we carry our own security analysis.

4.1 Formal framework

An asymmetric encryption scheme is *semantically secure* if no polynomial-time attacker can learn any bit of information about the plaintext from the ciphertext, except its length. More formally, an asymmetric encryption scheme is (t, ε)-IND, where IND stands for *indistinguishable*, if for any adversary $\mathcal{A} = (A_1, A_2)$ with running time bounded by t, the advantage

$$\mathsf{Adv}^{\mathsf{ind}}(\mathcal{A}) = 2 \times \Pr_{\substack{b \xleftarrow{R} \{0,1\} \\ r \xleftarrow{R} \Omega}} \left[\begin{array}{l} (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathcal{K}(1^k), (m_0, m_1, s) \leftarrow A_1(\mathsf{pk}) \\ c \leftarrow \mathcal{E}_{\mathsf{pk}}(m_b; r) : A_2(c, s) \stackrel{?}{=} b \end{array} \right] - 1$$

is $< \varepsilon$, where the probability space includes the internal random coins of the adversary, and m_0, m_1 are two equal length plaintexts chosen by the adversary in the message-space $\mathcal{M}$. Another security notion has been defined in the literature, called *non-malleability* [13]. Informally it states that it is impossible to derive, from a given ciphertext, a new ciphertext such that the plaintexts are meaningfully related. We won't discuss this notion any further since it has been proven equivalent to semantic security in an extended attack model.

The above definition of semantic security covers passive adversaries. It is a *chosen–plaintext* or CPA attack since the attacker can only encrypt plaintexts. In the extended model, the *adaptive chosen–ciphertext* or CCA attack, the adversary is given access to a decryption oracle and can ask the oracle to decrypt any ciphertext, with the only restriction that it should be different from the challenge ciphertext. It has been proven in [4] that, under CCA, semantic security and non-malleability are equivalent. This is the strongest security notion currently considered.

4.2 The Self-Evaluation Report

The self-evaluation report [18] includes a security analysis, which is extremely hard to read and presumably flawed. For example, we believe that lemma 2.1, page 12,

is incorrect. Firstly, we do not understand the mathematical argument supporting inequality (2): more precisely, we do not see the reason for having $s \neq s'$. Furthermore, the lemma does not cover situations where several plaintexts are possible. We believe that a correct version of lemma 2.1 should be derived from q_D invocations of the following estimate, which bounds the probability that a single decryption query is not properly simulated. In this estimate, $x^\star$, $r^\star$, $s^\star$ and $t^\star$ are related to the challenge, while x, r, s and t are used for the decryption query.

Lemma 1 *Under the assumption that $s^\star$ has not been queried from* H*,*

$$\Pr[\mathit{Fail}] \leq \frac{4(q_G+1)}{2^{k_0}} + \frac{4}{2^{k_1}}.$$

Proof. One should actually consider up to four pre-images yielding up to four pairs (r_i, s_i). To make things simpler, we consider only one such pair (r, s) and multiply the estimate by four. In other words, we bound the probability of the event Fail_i that the i-th candidate pair yields the adequate redundancy, while it is not available to the plaintext extractor, which produces a mismatch between the decryption oracle and the simulator. For the sake of clarity, we omit the i subscript and write $\Pr[\mathsf{Fail}]$ in place of $\Pr[\mathsf{Fail}_i]$. We bound $\Pr[\mathsf{Fail}]$ by:

$$\begin{aligned} & \Pr[\mathsf{Fail} \wedge \mathsf{AskR}] + \Pr[\mathsf{Fail} \wedge \neg\mathsf{AskR}] \\ \leq\ & \Pr[\mathsf{Fail} \wedge \mathsf{AskR} \wedge \mathsf{AskS}] + \Pr[\mathsf{Fail} \wedge \mathsf{AskR} \wedge \neg\mathsf{AskS}] \\ & + \Pr[\mathsf{Fail} \wedge \neg\mathsf{AskR}] \\ \leq\ & \Pr[\mathsf{AskR} \mid \neg\mathsf{AskS}] + \Pr[\mathsf{Fail} \wedge \neg\mathsf{AskR}]. \end{aligned}$$

Firstly, one easily sees that $\Pr[\mathsf{AskR} \mid \neg\mathsf{AskS}] \leq q_G/2^{k_0}$. Secondly,

$$\begin{aligned} \Pr[\mathsf{Fail} \wedge \neg\mathsf{AskR}] &= \Pr[\mathsf{Fail} \wedge r = r^\star \wedge \neg\mathsf{AskR}] \\ &\quad + \Pr[\mathsf{Fail} \wedge r \neq r^\star \wedge \neg\mathsf{AskR}] \\ &\leq \Pr[r = r^\star] \\ &\quad + \Pr[\mathsf{Fail} \mid \neg\mathsf{AskR} \wedge r \neq r^\star]. \end{aligned}$$

The first term means that $\mathsf{H}(s) \oplus t = r = r^\star = \mathsf{H}(s^\star) \oplus t^\star$, while $s^\star$ has not been queried from H. Such an even cannot happen with probability greater than 2^{-k_0}. The latter means that the redundancy holds, whereas $\mathsf{G}(r)$ is still undefined. This event occurs with probability 2^{-k_1}. Finally,

$$\Pr[\mathsf{Fail}] \leq \frac{q_G+1}{2^{k_0}} + \frac{1}{2^{k_1}},$$

and we just have to introduce a multiplicative factor 4, to take into account the fact that we only considered one candidate pair (r, s) instead of four. □

Provided the rest of the proof provided in [18] is correct, as it appears, the security result becomes:

Theorem 3 *Let $\mathcal{A}$ be a CCA–adversary $\mathcal{A}$ breaking the semantic security of the HIME(R) scheme, within time bound t, with advantage ε, making q_D, q_G and q_H queries to the decryption oracle and the hash functions G and H, respectively. Then there exists an adversary $\mathcal{B}$ factoring k-bit integer $N = p^d q$ with success probability ε' and within time bound t' where*

$$\begin{aligned} \varepsilon' &\geq \frac{1}{3} \times \left(\varepsilon - \frac{2q_G + 4q_D(q_G+1)}{2^{k_0}} - \frac{4q_D}{2^{k_1}} \right) \\ t' &\leq t + q_G q_H \cdot T_{\mathcal{E}}(k) + q_H \cdot T_{\mathsf{Cop}}(k,2) + T_{\mathsf{gcd}}(k), \end{aligned}$$

where $T_{\mathcal{E}}(k)$ denotes the computing time for encryption, $T_{\mathsf{Cop}}(k,d)$ the time needed for Coppersmith's algorithm to find small roots of a polynomial of degree d modulo a k-bit integer, and $T_{\mathsf{gcd}}(k)$ the time required for computing the gcd *of two k-bit integers*

4.3 Security Proof

We now perform our own security analysis, which we base on the machinery introduced by Shoup [26]. Although it is in a different framework, our proof uses simulations which are basically similar to those in [18].

We want to prove, in the random oracle, that HIME(R)is IND-CCA, based on the assumption that factoring is hard. More precisely, we would like to turn a CCA adversary $\mathcal{A}$ into a machine inverting f, and apply Theorem 2. However, as shown in [26], this strategy is actually hopeless. In place, we turn $\mathcal{A}$ into a machine that *partially inverts* f, i.e. finds a part of the preimage of a given value, leaving only k_0 trailing bits unknown. Using Coppersmith algorithm [11], and the assumption that $2k_0 < k$, this is enough to fully invert f, by recovering the missing bits.

We prove the following.

Theorem 4 *Let $\mathcal{A}$ be a CCA–adversary $\mathcal{A}$ against the semantic security of HIME(R), within time bound t, with advantage ε, making q_D, q_G and q_H queries to the decryption oracle and the hash functions G and H, respectively. Then:*

$$\varepsilon \leq 6 \times \mathsf{Succ}^{\mathsf{fact}}(k, t + q_G q_H \cdot T_{\mathcal{E}}(k) + q_H \cdot T_{\mathsf{Cop}}(k,2) + T_{\mathsf{gcd}}(k)) + \frac{q_G + 4q_D + 4q_G q_D}{2^{k_0 - 1}} + \frac{4q_D}{2^{k_1 - 1}}$$

where $T_{\mathcal{E}}(k)$ denotes the computing time for encryption, $T_{\mathsf{Cop}}(k,d)$ the time needed for Coppersmith's algorithm to find small roots of a polynomial of degree d, modulo a k-bit integer, and $T_{\mathsf{gcd}}(k)$ the time required for computing the gcd *between two k-bit long integers.*

The theorem follows from the next lemma.

Lemma 2 *Let $\mathcal{A}$ be a CCA–adversary $\mathcal{A}$ against the semantic security of HIME(R), within time bound t, with advantage ε, making q_D, q_G and q_H queries to the decryption oracle and the hash functions G and H, respectively. Then there exists an adversary $\mathcal{B}$ inverting f, with success probability ε' and within time bound t' where*

$$\begin{aligned} \varepsilon' &\geq \frac{\varepsilon}{2} - \frac{q_G + 4q_D + 4q_G q_D}{2^{k_0}} - \frac{4q_D}{2^{k_1}} \\ t' &\leq t + q_G q_H \cdot T_{\mathcal{E}}(k) + q_H \cdot T_{\mathsf{Cop}}(k, 2). \end{aligned}$$

4.3.1 Notations and Intuition

In the following, we use starred letters ($r^\star$, $s^\star$, $t^\star$ and $y^\star$) to refer to the challenge ciphertext, whereas unstarred letters (r, s, t and y) refer to the various ciphertexts queried from the decryption oracle.

As appears in the proof of lemma 1, we claim that r cannot match with $r^\star$, unless $s^\star$ is queried from H. This is because $r^\star = t^\star \oplus \mathsf{H}(s^\star)$ equals $r = t \oplus \mathsf{H}(s)$ with minute probability. Thus, if r is not queried, then $\mathsf{G}(r)$ is random and we similarly infer that the extractor can safely reject. The argument fails only if $s^\star$ is queried.

Thus, rejecting when it cannot combine elements of the lists of G and H so as to build a pre-image of y, the plaintext extractor is only wrong with minute probability, unless $s^\star$ has been queried by the adversary.

This seems to show that such an attack is impossible if it is difficult to "partially" invert f, which means finding s from $y = f(s\|t)$. As noted in [18], partial inversion of f easily leads to complete inversion, using Coppersmith's algorithm [11].

4.3.2 Formal Proof

As already stated, our method of proof is inspired by Shoup [26]: we define a sequence Game_1, Game_2, etc of modified attack games starting from the actual game Game_0. Each of the games operates on the same underlying probability space: the public and private keys of the cryptosystem, the coin tosses of the adversary $\mathcal{A}$, the random oracles G and H and the hidden bit b for the challenge. Only the rules defining how the view is computed differ from game to game. To go from one game to another, we repeatedly use the following lemma from [26]:

Lemma 3 *Let E_1, E_2 and F_1, F_2 be events defined on a probability space*

$$\Pr[\mathsf{E}_1 \wedge \neg\mathsf{F}_1] = \Pr[\mathsf{E}_2 \wedge \neg\mathsf{F}_2] \text{ and } \Pr[\mathsf{F}_1] = \Pr[\mathsf{F}_2] = \varepsilon \Rightarrow |\Pr[\mathsf{E}_1] - \Pr[\mathsf{E}_2]| \leq \varepsilon.$$

Proof. The proof follows from easy computations:

$$\begin{aligned} |\Pr[\mathsf{E}_1] - \Pr[\mathsf{E}_2]| &= |\Pr[\mathsf{E}_1 \wedge \neg\mathsf{F}_1] + \Pr[\mathsf{E}_1 \wedge \mathsf{F}_1] - \Pr[\mathsf{E}_2 \wedge \neg\mathsf{F}_2] - \Pr[\mathsf{E}_2 \wedge \mathsf{F}_2]| \\ &= |\Pr[\mathsf{E}_1 \wedge \mathsf{F}_1] - \Pr[\mathsf{E}_2 \wedge \mathsf{F}_2]| \end{aligned}$$

$$
\begin{aligned}
&= |\Pr[\mathsf{E}_1 \mid \mathsf{F}_1] \cdot \Pr[\mathsf{F}_1] - \Pr[\mathsf{E}_2 \mid \mathsf{F}_2] \cdot \Pr[\mathsf{F}_2]| \\
&= |\Pr[\mathsf{E}_1 \mid \mathsf{F}_1] - \Pr[\mathsf{E}_2 \mid \mathsf{F}_2]| \cdot \varepsilon \leq \varepsilon
\end{aligned}
$$

□

4.3.3 Simulating the Decryption Oracle.

In order to prove the security against adaptive chosen-ciphertext attacks, it is necessary to simulate calls to a decryption oracle. As usual, this goes through the design of a plaintext-extractor.

The latter receives as part of its input two lists of query-answer pairs corresponding to calls to the random oracles G and H, which we respectively denote by G-List and H-List. It also receives a valid ciphertext $y^\star$. Given these inputs, the extractor should decrypt a candidate ciphertext $y \neq y^\star$.

On query $y = f(s\|t)$, it inspects each query/answer pair $(\gamma, G_\gamma) \in \mathsf{G}$-List and $(\delta, H_\delta) \in \mathsf{H}$-List. For each combination of elements, one from each list, it defines

$$\sigma = \delta, \theta = \gamma \oplus H_\delta, \mu = G_\gamma \oplus \delta,$$

and checks whether $y = f(\sigma\|\theta)$ and whether μ is correctly formatted. If so, it outputs the n leading bits of μ and stops. If no such pair is found, the extractor returns a "Reject" message.

In exceptional cases, there may be several possible outputs for the plaintext-extractor, the same way decryption errors may formally appear with negligible probability (see theorem 2.1 [17, page 10]). This is related to the existence of four square roots for any quadratic residue y: with minute probability more than one candidate may show the adequate redundancy. Of course we can decide to treat the various square roots in a prescribed order, say in increasing order. This is not enough to avoid any resulting mismatch between the simulation and the decryption oracle. However, with this rule, this can only happen if the decryption oracle performs decryption using a candidate pair (r, s) which is not available for the plaintext extractor. As shown in Lemma 1, under the assumption that $s^\star$ has not been asked from H, the probability of this event is bounded by $(q_G + 1)/2^{k_0} + 1/2^{k_1}$, and since there are four candidate pairs, the probability that the plaintext extractor is wrong is bounded by $4(q_G+1)/2^{k_0}+4/2^{k_1}$

4.3.4 Semantic Security against Adaptive Chosen-Ciphertext Attacks.

In the following, $y^\star$ is the challenge ciphertext, obtained from the encryption oracle. Since we have in mind using the plaintext-extractor instead of the decryption oracle, trying to contradict semantic security, we assume that $y^\star$ is a ciphertext of m_b and denote by $r^\star$ its random seed. Thus, we have:

$$r^\star = \mathsf{H}(s^\star) \oplus t^\star \text{ and } \mathsf{G}(r^\star) = s^\star \oplus (m_b\|0^{k_1}).$$

Game_0**:** A pair of keys $(\mathsf{pk}, \mathsf{sk})$ is generated using $\mathcal{K}(1^k)$. Adversary A_1 is fed with pk, and outputs a pair of messages (m_0, m_1). Next a challenge ciphertext is produced by flipping a coin b and producing a ciphertext $y^\star$ of m_b. This ciphertext comes from a random $r^\star \stackrel{R}{\leftarrow} \{0,1\}^{k_0}$ and a string $x^\star$ such that $y^\star = f(x^\star)$. We set $x^\star = s^\star \,\|\, t^\star$, where $s^\star = (m_b \,\|\, 0^{k_1}) \oplus \mathsf{G}(r^\star)$ and $t^\star = r^\star \oplus \mathsf{H}(s^\star)$. On input $y^\star$, A_2 outputs bit b'.

Note that the adversary is given access to a decryption oracle $\mathcal{D}_{\mathsf{sk}}$ during both steps of the attack. The only requirement is that the challenge ciphertext $y^\star$ cannot be queried from the decryption oracle.

We denote by S_0 the event $b' = b$ and use a similar notation S_i in any Game_i below. By definition, we have

$$\Pr[S_0] = 1/2 + \varepsilon/2.$$

Game_1**:** We modify the above game, by making the value of the random seed $r^\star$ explicit and moving its generation upfront. In other words, one randomly chooses ahead of time, $r^+ \stackrel{R}{\leftarrow} \{0,1\}^{k_0}$ and $g^+ \stackrel{R}{\leftarrow} \{0,1\}^{k-k_0-1}$, and uses r^+ instead of $r^\star$, as well as g^+ instead of $\mathsf{G}(r^\star)$. The game obeys the following two rules:

Rule 1. $r^\star = r^+$ and $s^\star = (m_b \,\|\, 0^{k_1}) \oplus g^+$, from which it follows that

$$t^\star = r^\star \oplus \mathsf{H}(s^\star), x^\star = s^\star \,\|\, t^\star \text{ and } y^\star = f(x^\star);$$

Rule 2. whenever the random oracle G is queried at r^+, the answer is g^+.

Since we replace a pair of elements, $(r^\star, \mathsf{G}(r^\star))$, by another, (r^+, g^+), with exactly the same distribution:

$$\Pr[S_1] = \Pr[S_0].$$

Game_2**:** In this game, we drop the second rule above and restore (potentially inconsistent) calls to G. Therefore, g^+ is just used in $x^\star$ but does not appear in the computation. Thus, the input to A_2 follows a distribution that does not depend on b. Accordingly, $\Pr[S_2] = 1/2$.

One may note that Game_1 and Game_2 may differ if $r^\star$ is queried from G. Let AskG_2 denotes the event that, in Game_2, $r^\star$ is queried from G. This covers queries coming from the adversary or the decryption oracle, but not calls from the encryption oracle for producing the challenge. We will use an identical notation AskG_i for any Game_i below. Then

$$|\Pr[S_2] - \Pr[S_1]| \leq \Pr[\mathsf{AskG}_2].$$

Game_3**:** We now define $s^\star$ as an independent random variable, as well as $\mathsf{H}(s^\star)$. In other words, one randomly chooses ahead of time, $s^+ \xleftarrow{R} \{0,1\}^{k-k_0-1}$ and $h^+ \xleftarrow{R} \{0,1\}^{k_0}$, and uses s^+ instead of $s^\star$, as well as h^+ instead of $\mathsf{H}(s^\star)$. The only change is that $s^\star = s^+$ instead of $(m_b \,\|\, 0^{k_1}) \oplus g^+$. This implicitly defines g^+ from s^+ and b. The game uses the following two rules:

Rule 1'. $g^+ = (m_b \,\|\, 0^{k_1}) \oplus s^+$ and $t^\star = r^\star \oplus h^+$;

Rule 2'. whenever the random oracle H is queried at s^+, the answer is h^+.

Since we replace the quadruple $(s^\star, \mathsf{H}(s^\star), g^+, b)$ by another with exactly the same distribution:

$$\Pr[\mathsf{AskG}_3] = \Pr[\mathsf{AskG}_2].$$

Game_4**:** In this game, we drop the second rule above and restore (potentially inconsistent) calls to H. Therefore, h^+ is just used in $x^\star$ but does not appear in the computation. One may note that Game_3 and Game_4 may differ if $s^\star$ is queried from H. Let AskH_4 denote the event that, in Game_4, $s^\star$ is queried from H. This covers queries form the adversary, or the decryption oracle, but not calls from the encryption oracle for producing the challenge. We will use an identical notation AskH_i for any Game_i below. Then

$$|\Pr[\mathsf{AskG}_4] - \Pr[\mathsf{AskG}_3]| \leq \Pr[\mathsf{AskH}_4].$$

Furthermore, $r^\star = t^\star \oplus h^+$ is uniformly distributed, and independent of the adversary's view, since h^+ is never revealed: $\Pr[\mathsf{AskG}_4] \leq (q_G + 4q_D)/2^{k_0}$, where q_G and q_D denote the number of queries asked by the adversary to G, or to the decryption oracle, respectively.

Game_5**:** In order to evaluate AskH_4, we again modify the previous game. When manufacturing the challenge ciphertext, we simply set $y^\star = y^+$, ignoring the encryption algorithm altogether. The value y^+ is given as an auxiliary data, generated by $y^+ = f(x^+)$ for $x^+ \xleftarrow{R} \{0,1\}^{k-1}$, following the definition of f. Once again, the distribution of $y^\star$ remains the same. The previous method defining $y^\star = f(s^\star \| t^\star)$, with $s^\star = s^+$ and $t^\star = h^+ \oplus r^+$ was already generating the input of f with a uniform distribution over the $k-1$-bit elements. Thus, we have:

$$\Pr[\mathsf{AskH}_5] = \Pr[\mathsf{AskH}_4].$$

We now deal with the decryption oracle, which has remained perfect up to this game.

Game_6: We make the decryption oracle reject all ciphertexts y for which it meets at decryption time an r value not been previously queried from G by the adversary. Since there are at most four square roots x of y in the appropriate range, four such values of r may appear. This makes a difference only if y is a valid ciphertext, while $\mathsf{G}(r)$ has not been asked. Since $\mathsf{G}(r)$ is uniformly distributed, $\mathsf{G}(r) \oplus s$ satisfies the required redundancy with probability less than $1/2^{k_1}$. Summing up for all decryption queries and all candidate values for r, we get

$$|\Pr[\mathsf{AskH}_6] - \Pr[\mathsf{AskH}_5]| \leq \frac{4q_D}{2^{k_1}}.$$

Game_7: We now make the decryption oracle reject all ciphertexts y for which it meets at decryption time an s value not been previously queried from H by the adversary. This makes a difference only if y is a valid ciphertext, and $\mathsf{H}(s)$ has not been asked, while the corresponding $\mathsf{G}(r)$ has been queried (otherwise it has already been rejected in the previous game). Since $r = \mathsf{H}(s) \oplus t$ is uniformly distributed, it has been queried from G by the aversary, with probability less than $q_G/2^{k_0}$. Summing up for all decryption queries and all candidate values for s, we get

$$|\Pr[\mathsf{AskH}_7] - \Pr[\mathsf{AskH}_6]| \leq 4q_D \times \frac{q_G}{2^{k_0}}.$$

Game_8: We finally replace the decryption oracle by the plaintext-extractor which perfectly simulates the decryption, since all r and s values needed for decryption have been previously queried:

$$\Pr[\mathsf{AskH}_8] = \Pr[\mathsf{AskH}_7].$$

If AskH_8 happens, then, for at least one element s queried to H, there exists $t \in \{0,1\}^{k_0}$ such that $(t + s \times 2^{k_0})^2 = y^+ \bmod N$.

Trying elements s queried from H during this game, one can extract a square root of y^+, smaller than 2^{k-1}, by using Coppersmith's algorithm [11]:

$$\Pr[\mathsf{AskH}_9] \leq \mathsf{Succ}^{\mathsf{ow}}_f(t'' + q_H T_{\mathsf{Cop}}(k, 2)),$$

where t'' is the running time of Game_8.

Therefore,

$$\begin{aligned}
\frac{\varepsilon}{2} &= |\Pr[S_0] - \Pr[S_2]| \leq \Pr[\mathsf{AskG}_2] \leq \Pr[\mathsf{AskG}_4] + \Pr[\mathsf{AskH}_4] \\
&\leq \frac{q_G + 4q_D}{2^{k_0}} + \Pr[\mathsf{AskH}_5] \leq \frac{q_G + 4q_D}{2^{k_0}} + \frac{4q_D}{2^{k_1}} + \Pr[\mathsf{AskH}_6] \\
&\leq \frac{q_G + 4q_D}{2^{k_0}} + \frac{4q_D}{2^{k_1}} + 4q_D \times \frac{q_G}{2^{k_0}} + \Pr[\mathsf{AskH}_7] \\
&\leq \frac{q_G + 4q_D + 4q_D q_G}{2^{k_0}} + \frac{4q_D}{2^{k_1}} + \mathsf{Succ}^{\mathsf{ow}}_f(t').
\end{aligned}$$

To conclude the proof of Lemma 2, it remains to comment on the running time $t' = t'' + q_H T_{\mathsf{Cop}}(k, 2)$. Although the plaintext-extractor is called q_D times, there is no q_D multiplicative factor in the bound for t''. This comes from a simple book-keeping argument. Instead of only storing the lists G-List and H-List, one stores an additional structure consisting of tuples $(\gamma, G_\gamma, \delta, H_\delta, y)$. A tuple is included only for $(\gamma, G_\gamma) \in$ G-List and $(\delta, H_\delta) \in$ H-List. For such a pair, one defines $\sigma = \delta$, $\theta = \gamma \oplus H_\delta$, $\mu = G_\gamma \oplus \delta$, and computes $y = f(\sigma, \theta)$. If the redundancy holds for μ, one stores the tuple $(\gamma, G_\gamma, \delta, H_\delta, y)$. The cumulative cost of maintaining the additional structure is $q_G q_H T_{\mathcal{E}}(k)$ but, handling it to the plaintext-extractor allows to output the expected decryption of y, by table lookup, in constant time. Of course, a time-space tradeoff is possible, giving up the additional table, but raising the computing time to $q_D q_G q_H T_{\mathcal{E}}(k)$.

4.4 Alternative Proof

We now propose an alternative security analysis, which leads to the following exact security result.

Theorem 5 *Let $\mathcal{A}$ be a CCA–adversary $\mathcal{A}$ against the semantic security of HIME(R), within time bound t, with advantage ε, making q_D, q_G and q_H queries to the decryption oracle and the hash functions* G *and* H*, respectively. Then,*

$$\varepsilon \leq 6 \times \mathsf{Succ}^{\mathsf{fact}}(k, t + q_H \cdot (q_D + 1) \cdot T_{\mathsf{Cop}}(k, 2) + T_{\mathsf{gcd}}(k)) + \frac{q_G + 4q_D + 4q_G q_D}{2^{k_0 - 1}} + \frac{4q_D}{2^{k_1 - 1}}.$$

Note that the result differs from the previous only in terms of the time complexity.

Our alternative proof differs from the previous one in the decryption simulation, for which we use Coppersmith's algorithm [11]. The new plaintext-extractor still receives as part of its input two lists of query/answer pairs corresponding to calls to the random oracles G and H, which we respectively denote by G-List and H-List. It also receives a valid ciphertext $y^\star$. Given these inputs, the extractor attempts to decrypt a candidate ciphertext $y \neq y^\star$.

On query $y = f(s\|t)$, it inspects each query/answer pair $(\sigma, H_\sigma) \in$ H-List. For each σ, using Coppersmith's algorithm, it computes all $\theta \in \{0,1\}^{k_0}$ such that $(\theta + \sigma \times 2^{k_0})^2 = y \bmod N$. It then defines

$$s = \sigma, t = \theta, r = t \oplus H_\sigma, M = \mathsf{G}(r) \oplus s,$$

and checks whether the redundancy holds. If for some s, the redundancy holds for M, the plaintext-extractor returns the n leading bits of M. Otherwise, it returns a "Reject" message.

Comments. Once again, in exceptional cases, there may be several possible outputs for the plaintext-extractor. Again, one can decide to treat the various square roots in a prescribed order. Rererring to the proof in section 4.3.4, we see that the Game_6 and Game_7 duly eliminate, as before, all cases where a mismatch could appear between the plaintext extractor and the decryption oracle. The cost of the simulation is linear in q_H only, which saves a q_G factor. However, the bookkeeping arguments that allowed to avoid the q_D factor is lost. On the other hand, no additional table is required. Therefore, the new reduction has memory requirement $\mathcal{O}(q_G + q_H)$, instead of $\mathcal{O}(q_G q_H)$ in the reductions proposed in [18] and above in the present report.

4.5 Practical Security Estimates

We now try to understand whether the figures shown above can be used to derive meaningful estimates for practical parameters, based on the time an adversary could spend on breaking semantic security. As in the specification, we set $k_0 = 128$ and $k_1 = 128$, $d = 2$, and try to obtain a lower bound on $k \geq 1024$. Following many authors, we take the usual values for q_G, q_H and q_D:

$$q_G \sim q_H \sim 2^{60} \text{ and } q_D \sim 2^{30},$$

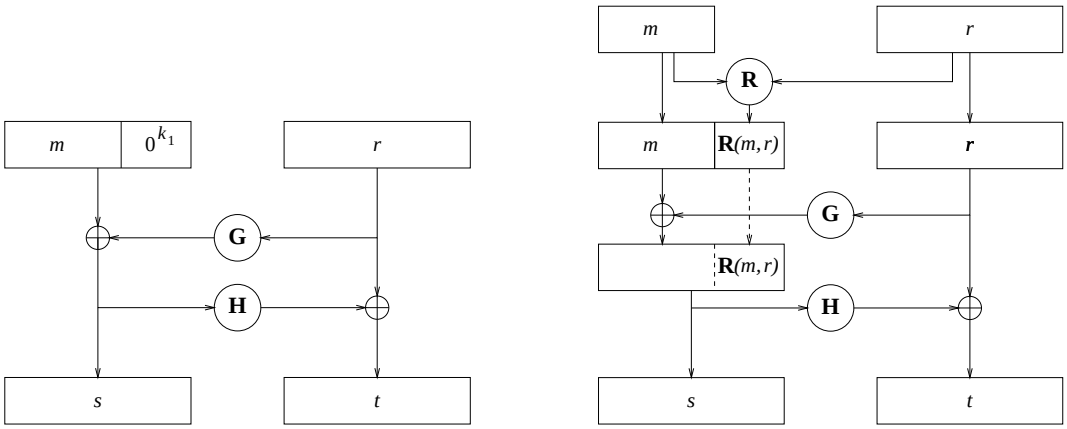
and we obtain that such a "standard" adversary could be used for factoring with success probability ε', within a time bound t' where

$$\begin{aligned} \varepsilon' &\geq \frac{1}{6} \times \left(\varepsilon - \frac{2^{60} + 4 \cdot 2^{30} + 4 \cdot 2^{90}}{2^{127}} - \frac{4 \cdot 2^{30}}{2^{127}} \right) \geq \frac{\varepsilon}{6} - \frac{1}{2^{37}} \\ t' &\leq t + 2^{90} T_{\mathsf{Cop}}(k, 2) \leq t + 2^{100} k^3. \end{aligned}$$

For this time upper-bound, we assumed $T_{\mathsf{Cop}}(k, 2) \leq 1000 \cdot k^3$. Finally, an adversary able to learn one bit with advantage $1/2$, within time $2^{100} k^3$, can be used to factor n within less than $2^{101} k^3$.

In the table below, we compare the factoring time of the reduction with those coming from the estimates for the complexity of ECM and NFS, $C_{ECM}(|p|)$ and $C_{NFS}(k)$, as reported in Section 3.2. The last column provides the size n of the plaintext. Note that the size of the ciphertext is k.

$\lvert p \rvert$	k	$\log C_{ECM}(\lvert p \rvert)$	$\log C_{NFS}(k)$	$101 + 3 \log k$	n
341	1024	81	85	131	768
448	1344	86	96	134	1088
512	1536	93	102	134	1280
682	2048	110	115	134	1792
1024	**3072**	**139**	**137**	**137**	**2816**



OAEP padding OAEP+ padding

Figure 2: OAEP and OAEP+ Paddings

Thus, if one believes that ECM or NFS are best possible, the reduction suggests to set parameter k above 3072. This shows that the reduction only provides a qualitative assurance that the scheme is not flawed and that it cannot be interpreted with the suggested parameters.

5 Comparison with other Schemes

5.1 RSA–OAEP

Figure 2, shows the OAEP construction, which can be used in conjunction with the RSA function [24]. On this picture, r is a random string of bit-length k_0, m is a message of bit-length n, and k_1 is the bit-length of the redundancy, of the message. The bit-length of the modulus is $k = n + k_0 + k_1 + 1$, and k is significantly larger than k_0, say $k > 2k_0$. Under these assumptions, the following theorem is proven in [16]:

Theorem 6 *Let $\mathcal{A}$ be a CCA–adversary against the semantic security of RSA–OAEP, with running time bounded by t and advantage ε, making q_D, q_G and q_H queries to the decryption oracle, and the hash functions G and H respectively. Then, the RSA problem can be solved with probability ε' greater than*

$$\frac{\varepsilon^2}{4} - \varepsilon \cdot \left(\frac{2q_D q_G + q_D + q_G}{2^{k_0}} + \frac{2q_D}{2^{k_1}} + \frac{32}{2^{k-2k_0}} \right)$$

within time bound $t' \leq 2t + q_H \cdot (q_H + 2q_G) \times \mathcal{O}(k^3)$.

It can be noted that the probability of success that the reduction is worse than in theorem 5. Firstly, the success probability ε' is quadratic in ε. More importantly, the time complexity is quadratic in the number of queries to the random oracles. We can use the estimate $121 + 3\log k$ for the (logarithmic) time complexity derived from a "standard" adversary.

k	$\log C_{NFS}(k)$	$121 + 3\log k$	n
1024	85	151	768
1344	96	152	1088
1536	102	153	1280
2048	115	154	1792
3072	137	155	2816
4096	**155**	**157**	**3840**

Interpreting the table above, as was done in section 4.5, would thus lead to larger moduli.

5.2 RSA–OAEP+

On Figure 2, the OAEP+ construction [26] is shown. This construction can be used in conjunction with the RSA function [24]. On this picture, r is a random string of bit-length k_0, m is a message of bit-length n, and R is a randomly looking function outputting k_1 bits of redundancy. The bit-length of the modulus is $k = n + k_0 + k_1 + 1$. Under these assumptions, the following theorem is proven in [26]:

Theorem 7 *Let $\mathcal{A}$ be a CCA–adversary against the semantic security of RSA–OAEP+, with running time bounded by t and advantage ε, making q_D, q_G, q_H and q_R queries to the decryption oracle, and the hash functions* G, H *and* R *respectively. Then, the RSA problem can be solved with probability ε' greater than*

$$\frac{\varepsilon}{2} - \frac{q_D + q_R}{2^{k_1}} - \frac{(q_D + 1) \cdot q_G}{2^{k_0}}$$

within time bound $t' \leq t + q_G q_H \cdot T_{\mathcal{E}}(k)$.

Here, the probability ε' is much tighter. However, the reduction remains quadratic in the number of queries to the random oracles. Thus, the situation is essentially similar to the case of RSA–OAEP.

5.3 RSA–SAEP and Rabin–SAEP

Figure 3 shows the SAEP construction from [7], which can be used in conjunction both with the RSA function [24], and the Rabin squaring function [23], also used in

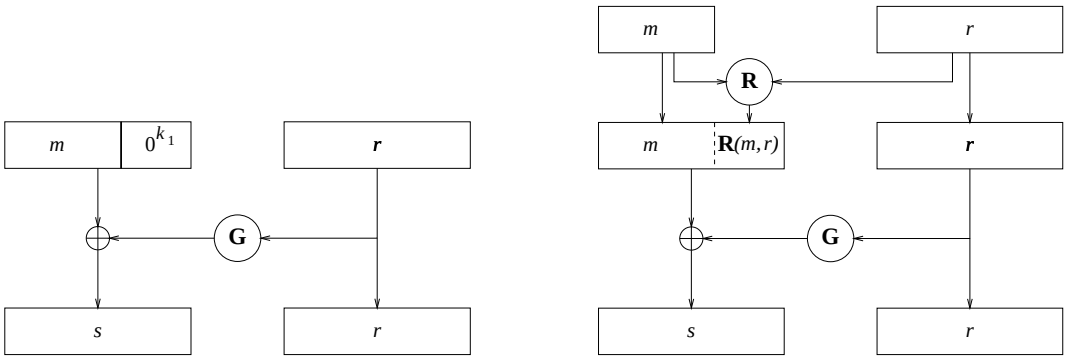


Figure 3: SAEP and SAEP+ Paddings

HIME(R). On this picture, r is a random string of bit-length k_0, m is a message of bit-length n, and k_1 is the bit-length of the redundancy. The bit-length of the modulus is $k = n+k_0+k_1+2$, where k is significantly larger than k_0, say $k > 4n$ and $k > 2(n+k_1)$. Under these assumptions, the following theorem is proven in [7]:

Theorem 8 *Let $\mathcal{A}$ be a CCA–adversary against the semantic security of Rabin–SAEP, with running time bounded by t and advantage ε, making q_D and q_G queries to the decryption oracle, and the hash function G respectively. Then, RSA moduli can be factored with probability ε' greater than*

$$\frac{\varepsilon}{6} \times \left(1 - \frac{2q_D}{2^{k_1}} - \frac{2q_D}{2^{k_0}}\right)$$

within time bound $t' \leq t + q_D q_H T_{\mathsf{Cop}}(k,2) + q_D T_{\mathsf{Cop}}(k,4)$.

The reduction is thus similar to the one provided for HIME(R), hence the table:

k	$\log C_{NFS}(k)$	$101 + 3\log k$	n
1024	85	131	256
1344	96	134	336
1536	102	134	384
2048	115	134	512
3072	**137**	**137**	**768**

Note that the decryption algorithm is less efficient than the corresponding algorithm of HIME(R). Furthermore, the restriction on the size of the message is quite strong.

A similar result can be proven for RSA–SAEP, with exponent $e = 3$ only. The reduction is relative to the RSA problem, and the following size constraints are needed: $k > 9n$ and $k > 3(n + k_1)$, which are worse than before.

5.4 RSA–SAEP+ and Rabin–SAEP+

Figure 3 shows the SAEP+ construction from [7], which can be used in conjunction both with the RSA function [24], and the Rabin squaring function [23], also used in HIME(R). On this picture, r is a random string of bit-length k_0, m is a message of bit-length n, and R is a randomly looking function outputting k_1 bits of redundancy. The bit-length of the modulus is $k = n + k_0 + k_1 + 2$, where k is significantly larger than k_0, say $k > 2(n + k_1)$. Under these assumptions, the following theorem is proven in [7]:

Theorem 9 *Let $\mathcal{A}$ be a CCA–adversary against the semantic security of Rabin–SAEP+, with running time bounded by t and advantage ε, making q_D, q_G and q_R queries to the decryption oracle, and the hash functions G and R respectively. Then, RSA moduli can be factored with probability ε' greater than*

$$\frac{\varepsilon}{6} \times \left(1 - \frac{q_D}{2^{k_1}} - \frac{q_D}{2^{k_0}}\right)$$

within time bound $t' \leq t + q_H T_{\mathsf{Cop}}(k, 2)$.

The reduction is very efficient, hence the table:

k	$\log C_{NFS}(k)$	$71 + 3\log k$	n
1024	85	101	384
1344	96	104	544
1536	102	104	640
2048	**115**	**104**	**896**

Note that the restriction on the size of the message is still quite strong.

There is a similar security result for RSA–SAEP+, with identical restrictions on the size of the message, but without any restriction on the exponent e:

Theorem 10 *Let $\mathcal{A}$ be a CCA–adversary against the semantic security of RSA–SAEP+, with running time bounded by t and advantage ε, making q_D, q_G and q_R queries to the decryption oracle, and the hash functions G and R respectively. Then, the RSA problem can be performed with probability ε' greater than*

$$\varepsilon^2 \times \left(1 - \frac{2q_D}{2^{k_1}} - \frac{2q_D}{2^{k_0}}\right)$$

within time bound $t' \leq 2t + q_H^2 T_{\mathcal{E}}(k)$.

Again, the reduction is quadratic in the number of queries asked to the random oracles. This can be interpreted as suggesting moduli with at least 4096 bits.

5.5 EPOC-2

EPOC-2 [22], uses moduli of the same form as in HIME(R), $N = p^2q$, and its security is similarly based on factoring. More precisely, let $\mathcal{A}$ be a CCA–adversary against the semantic security of EPOC-2, with running time bounded by t and advantage ε, making q_D, q_G and q_H queries to the decryption oracle, and the hash functions G and H respectively. Then, integers of the form $N = p^2q$ can be factored with probability approximately $\varepsilon/2$ within time bound $t + (q_G + q_H) \times \mathcal{O}(k^3)$. The reduction here is quite tight.

$\lvert p\rvert$	k	$\log C_{ECM}(\lvert p\rvert)$	$\log C_{NFS}(k)$	$61 + 3\log k$
341	1024	81	85	91
512	1536	93	102	94
682	**2048**	**110**	**115**	**94**

Note however that EPOC-2 is a hybrid scheme, and, as such, cannot be directly compared to the other public key encryption schemes.

5.6 Summary

The following table summarizes our findings. It shows the practical values of the bit length of the modulus suggested by the designers, as well as the theoretical value derived by reduction from the "standard" adversary, as above.

Scheme	Problem	Reduction cost (approx.)	suggested bitsize k	theoretical bitsize k
HIME(R)	$\mathsf{Fact}(p^2q)$	$q_D q_H \times \mathcal{O}(k^3)$	1344	3072
RSA–OAEP	$\mathsf{RSA}(pq, e)$	$2q_G q_H \times \mathcal{O}(k^3)$	1024	4096
RSA–OAEP+	$\mathsf{RSA}(pq, e)$	$q_G q_H \times \mathcal{O}(k^2)$	1024	4096
Rabin–SAEP	$\mathsf{Fact}(pq)$	$q_D q_H \times \mathcal{O}(k^3)$	1024	3072
Rabin–SAEP+	$\mathsf{Fact}(pq)$	$q_H \times \mathcal{O}(k^3)$	1024	2048
EPOC-2	$\mathsf{Fact}(p^2q)$	$(q_G + q_H) \times \mathcal{O}(k^3)$	1152	2048

We also include another table, that shows the respective time complexity $T_{\mathcal{E}}(k)$ and $T_{\mathcal{D}}(k)$ of encryption and decryption, measured in number of modular multiplications of k-bit integers. In this table, we have set $e = 2^{16} + 1$ for the RSA public exponent, and we have chosen 128 bit random values for EPOC-2. The table also shows the message size n, and the size c of the ciphertext.

Scheme	$T_{\mathcal{E}}(k)$	$T_{\mathcal{D}}(k)$	n	c
HIME(R)	1	$3k/27$	k-256	k
RSA–OAEP	17	$3k/8$	k-256	k
RSA–OAEP+	17	$3k/8$	k-256	k
Rabin–SAEP	1	$3k/8$	$k/4$	k
Rabin–SAEP+	1	$3k/8$	$(k-128)/4$	k
EPOC-2	384	124	any n	$k+n$

In terms of complexity, the three best schemes appear to be HIME(R), Rabin–SAEP+ and EPOC-2. They all rely on factoring. HIME(R) and Rabin–SAEP+ have a very efficient encryption algorithm, and HIME(R) and EPOC-2 have a very efficient decryption algorithm. In terms of bandwith, one notes that Rabin–SAEP+ limits the size of the plaintext to $k/4$, whereas EPOC-2 provides an hybrid encryption scheme, which can encrypt plaintext of any length, with a constant overhead of k bits. As for HIME(R), it can encrypt up to $k-256$ bit into a k-bit ciphertext. In a way, HIME(R) achieves one of the best performances. Note however that, as stated in section 4.5, the security analysis does not formally support a 1344 bit modulus.

6 Conclusion

Based on our security analysis, we believe that the public key cryptosystem HIME(R) has provable security, based on factoring, and is presumably secure with the proposed parameters. Furthermore, it ranks extremely well, in terms of efficiency, compared to similar schemes.

On the other hand, we are in an uncomfortable situation to strongly recommend the scheme, since we found the security proof provided by the designers rather unconvincing.

References

[1] D. Atkins, M. Graff, A. K. Lenstra, and P. Leyland. The Magic Words are Squeeming Ossifrage. Asiacrypt '94, Lecture Notes in Computer Science 917, (1995), pages 263–277.

[2] F. Almgren, G. Andersson, T. Granlund, L. Ivansson and S. Ulfberg. Singh challenge 10.
`http://wwwhome.cs.utwente.nl/ crypto/challenge-10.html`

[3] F. Bahr, J. Franke, and T. Kleinjung. Factorization of c158.
`http://www.crypto-world.com/announcements/c158.txt`

[4] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among Notions of Security for Public-Key Encryption Schemes. Crypto '98, Lecture Notes in Computer Science 1462, (1998), pages 26–45.

[5] M. Bellare and P. Rogaway. Optimal Asymmetric Encryption – How to Encrypt with RSA. Eurocrypt '94, Lecture Notes in Computer Science 950, (1995), pages 92–111.

[6] D. Boneh, G. Durfee, and N. Howgrave-Graham. Factoring $N = p^r q$ for large r, Crypto '99, Lecture Notes in Computer Science 1666, (1999), pages 326–337.

[7] D. Boneh. Simplified OAEP for the RSA and Rabin Functions. Crypto '01, Lecture Notes in Computer Science 2139, (2001), pages 275–291.

[8] R. P. Brent. Some Parallel Algorithms for Integer Factorisation. Euro-Par '99, Lecture Notes in Computer Science 1685, (1999), 1–22.

[9] S. Cavallar, B. Dodson, A. K. Lenstra, P. Leyland, W. Lioen, P. L. Montgomery, B. Murphy, H. te Riele, and P. Zimmermann. Factorization of RSA-140 using the Number Field Sieve. Asiacrypt '99, Lecture Notes in Computer Science 1716 (1999), pages 195–207.

[10] S. Cavallar, B. Dodson, A. K. Lenstra, W. Lioen, P. L. Montgomery, B. Murphy, H. te Riele, K. Aardal, J. Gilchrist, G. Guillerm, P. C. Leyland, J. Marchand, F. Morain, A. Muffett, C. Putnam, C. Putnam, and P. Zimmermann. Factorization of a 512-Bit RSA Modulus. Eurocrypt '00, Lecture Notes in Computer Science 1807 (2000), pages 1–18.

[11] D. Coppersmith. Finding a Small Root of a Univariate Modular Equation. Eurocrypt '96, Lecture Notes in Computer Science 1070, (1996), pages 155–165.

[12] D. Coppersmith. Finding a Small Root of a Bivariate Integer Equation; Factoring with High Bits Known. Eurocrypt '96, Lecture Notes in Computer Science 1070, (1996), pages 178–189.

[13] D. Dolev, C. Dwork, and M. Naor. Non-Malleable Cryptography. Proc. of the 23rd STOC. ACM Press, New York, 1991.

[14] R.-M. Elkenbracht-Huizing. An Implementation of the Number Field Sieve. *Exp. Math.* 5, (1996), pages 231-253.

[15] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA–OAEP is Secure under the RSA Assumption. Crypto '01, Lecture Notes in Computer Science 2139, (2001), pages 260–274.

[16] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA–OAEP is Secure under the RSA Assumption. Journal of Cryptology, (2002), to appear.

[17] Specification of HIME(R) Cryptosystem, Hitachi Ltd. (2001), `http://www.sdl.hitachi.co.jp/crypto/hime/`.

[18] Self Evaluation Report, HIME(R) Cryptosystem, Hitachi Ltd. (2001), `http://www.sdl.hitachi.co.jp/crypto/hime/`.

[19] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring Polynomials with Rational Coefficients. *Mathematische Ann.*, 261, (1982), pages 513–534.

[20] N. Lygeros, M. Mizony, and P. Zimmermann. A New ECM Record with 54 Digits. `http://www.desargues.univ-lyon1.fr/home/lygeros/Mensa/ecm54.html`.

[21] P. L. Montgomery. A Block Lanczos Algorithm for Finding Dependencies over GF(2). Eurocrypt '95, Lecture Notes in Computer Science 921, (1995), pages 106–120.

[22] T. Okamoto, S. Uchiyama, and E. Fujisaki. EPOC: Efficient Probabilistic Public-Key Encryption. Submission to IEEE P1363, ISO, CRYPTREC (1998).

[23] M. O. Rabin. Digitalized Signatures. *Foundations of Secure Computation*, Academic Press, New York, (1978), pages 155–166.

[24] R. L. Rivest, A. Shamir, and L. M. Adleman. Cryptographic Communications System and Method. US patent 4 405 829, September 20, 1983 (filed 14/12/1977).

[25] RSA Laboratories, Information on the RSA Challenge. `http://www.rsa.com/rsalabs/html/challenges/html`.

[26] V. Shoup. OAEP Reconsidered. Crypto '01, Lecture Notes in Computer Science 2139, (2001), pages 239–259.

[27] T. Takagi. Fast RSA-Type Cryptosystem Modulo $p^k q$. Crypto '98, Lecture Notes in Computer Science 1462, (1998), pages 318-326.