Evaluation of Security Level of Cryptography: The Revised Version of EPOC-2

Alfred Menezes
University of Waterloo
Contact: ajmeneze@uwaterloo.ca

December 14, 2001

Contents

1 Executive Summary

This report evaluates the EPOC-2 public-key encryption scheme as specified in [19]. The goals of EPOC-2 are to provide confidentiality and data integrity for messages of arbitrary lengths. The sender of encrypted data is assured that no one but the intended receiver can recover the plaintext, while the receiver is assured that the plaintext was not modified during transmission. EPOC-2 is provably secure in the random oracle model under the assumption that factoring integers of the form p^2q is intractable. We compare EPOC-2 to RSA-OAEP as specified in PKCS#1 v2.1 [25], and conclude that the advantages of EPOC-2 over RSA-OEAP do not seem to outweigh its comparative disadvantages.

2 Introduction

HISTORY OF EPOC-2. The EPOC-2 public-key encryption scheme combines the 1998 basic public-key encryption scheme of T. Okamoto and S. Uchiyama [21] and the 1999 padding scheme of E. Fujisaki and T. Okamoto [6]. The latest version of EPOC-2 [19] includes modifications which address some observations made by Shoup [26]. Throughout this document, we refer only to this revised version of EPOC-2.

COMPARING EPOC-2 TO OTHER ENCRYPTION SCHEMES. Statements regarding the security and performance of EPOC-2 are more meaningful when they are compared to the security and performance of other encryption schemes. EPOC-2 can be compared to any public-key encryption scheme, but it is advisable to compare it to some accepted, well-known and widely-used scheme. In addition, it should be compared to a scheme that has somewhat similar characteristics, and that would be appropriate for similar applications. The most suitable such scheme is RSA-OAEP (Optimal Asymmetric Encryption Padding) as specified in [25].

ORGANIZATION. The remainder of this report is organized as follows. In Section 3, we describe the EPOC-2 key generation, encryption and decryption procedures, and explain why the decryption procedure works. The security of EPOC-2 is analyzed in Section 4. Here we consider the provable security aspects of EPOC-2, the hardness of the underlying integer factorization problem, sizes of domain parameters, and implementation attacks. In Section 5, we consider performance issues such as encryption and decryption times, the use of EPOC-2 in some security protocols, and stream processing. Our conclusions are stated in Section 6.

3 EPOC-2 Specification

3.1 EPOC-2 Domain Parameters

The EPOC-2 scheme as described in [19] has certain parameters that (i) an encryptor and decryptor must agree upon and mutually support (for interoperability); and (ii) require the support of advanced implementation techniques. These parameters are typically fixed for a *domain* of intercommunicating users, and should be carefully selected since they affect both security and performance.

1. An essential domain parameter is $k = pLen$, the bit size of the prime factors used in the private key. This parameter is needed for both encryption and decryption.

2. Another domain parameter is the symmetric cipher S. The recommended symmetric cipher is Camellia which has key sizes of 128, 192 and 256 bits; other symmetric ciphers have not been explicitly prohibited. The encryptor and decryptor must agree on which symmetric cipher is to be used, and they both must support this cipher. For a block cipher such as Camellia, the mode of block cipher operation, such as ECB, CFB, CBC, or counter mode, must also be agreed upon and supported.

3. Another possible domain parameter involves a pair of auxiliary functions K and E. The function K is a *key derivation function*. It takes a $(k-1)$-bit input and outputs a bit string of the same bitlength as keys for S. The EPOC-2 specification recommends the key derivation function KDF2. The function E is a *mask generation function*[1]. It takes an input of arbitrary length and outputs a bitstring of length $l = 2k + v$, where v is also a domain parameter. The EPOC-2 specification recommends the mask generation function MGF1.

   The key derivation function KDF2 and mask generation function MGF1 are both constructed from a hash function. Their outputs are formed by concatenating the values produced by hashing the input together with a 32-bit counter that is incremented for each hash operation. The EPOC-2 specification recommends the hash function SHA-1 [16]. Since there is only one recommended hash function, it is not necessary to include any information about K and E as part of the domain parameters. However, if the specification is to allow upgrades to more secure hash functions such as SHA-256, SHA-384 and SHA-512 [17], then a hash function identifier should be included as a domain parameter.

The establishment or conveyance of domain parameters is generally outside the scope of a public-key encryption scheme. The EPOC-2 specification, however, explicitly includes the domain parameter k in the public key. This particular domain parameter is therefore explicitly

[1]This terminology is taken from the OAEP specification but seems inappropriate for EPOC-2 since nothing is "masked".

conveyed. Note that including k in the public key is redundant since it is equal to one-third of the bitsize of n.

3.2 EPOC-2 Key Pair Generation

To generate a public key pair for the domain parameter k, an entity A does the following:

1. Randomly select two distinct primes p and q, each of bitsize k.
2. Compute $n = p^2 q$.
3. Select an integer g uniformly at random from the interval $[1, n-1]$ until $\gcd(g, n) = 1$ and $g^{p-1} \bmod p^2$ has order p modulo p^2.
4. Select an integer h_0 uniformly at random from $[1, n-1]$ such that $\gcd(h_0, n) = 1$.
5. Compute $h = h_0^n \bmod n$.
6. Compute $g_p = g^{p-1} \mod p^2$.
7. Compute $L = (g_p - 1)/p$.
8. A's public key is (n, g, h, k).
9. A's private key is (p, q, k, L).

NOTES.

1. It should be clearly stated in the EPOC-2 specification that the primes p and q need to be generated uniformly at random from the set of all k-bit primes.
2. In step 3, an integer $g \in_R [1, n-1]$ has overwhelming probability that $\gcd(g, n) = 1$ and $g^{p-1} \bmod p^2$ has order p modulo p^2.

3.3 EPOC-2 Encryption

To encrypt a message m of arbitrary length for an entity A with public key (n, g, h, k), an entity B does the following:

1. Select random $r \in \{0, 1\}^{k-1}$.
2. Compute $d = S_{K(r,P)}(m)$. (P are "encoding parameters").
3. Compute $e = E(m \| r \| d \| P)$.
4. Compute $c = g^r h^e \bmod n$.
5. Output ciphertext (c, d).

NOTES.

1. Shamir's trick (Algorithm 14.88 in [15]) can be used to speed up the two exponentiations in step 4.
2. The variable names chosen in the EP-OU, ES-EPOC-2 and EME3 routines in the EPOC specification [19] are very confusing. For example, "f" is used to denote the octet string representation of "R" in EME3, while "r" is used to denote the octet string representation of "H" in EME3.
3. There is a minor error in the description of the variables f and r in the EME3 routine in [19]—f is the random value, and r is the message representation. A similar statement holds for the variables m and r in the EP-OU routine in [19].
4. The EPOC-2 specification [19] does not provide any guidance on the use of the "encoding parameters" P.

3.4 EPOC-2 Decryption

To decrypt a ciphertext (c,d), an entity A with private key (p,q,k,L) does the following:

1. Verify that $0 \le c \le n-1$; if not then output "invalid" and stop.
2. Compute $r = \big(\big((c^{p-1} \bmod p^2) - 1\big)/p\big)/L \bmod p$.
3. Verify that $r \in \{0,1\}^{k-1}$; if not then output "invalid" and stop.
4. Compute $m = S^{-1}_{K(r,P)}(d)$.
5. Compute $e = E(m \,\|\, r \,\|\, d \,\|\, P)$.
6. Verify that $c \equiv g^r h^e \pmod{q}$; if not then output "invalid".

NOTES.

1. Unlike EPOC-2 encryption, EPOC-2 decryption never uses the modulus n. Instead it uses the three smaller moduli p^2, p and q.
2. Clearly e can be reduced modulo $q-1$ before exponentiating in step 6.
3. Shamir's trick (Algorithm 14.88 in [15]) can be used to speed up the two exponentiations in step 6.
4. Powers of g and h can be precomputed to speed up the exponentiations in step 6.

3.5 Proof that EPOC-2 Decryption Works

It is not immediately clear from the encryption and decryption algorithms why EPOC-2 decryption successfully recovers the plaintext from a ciphertext generated by EPOC-2 encryption. For completeness, we provide a brief explanation of how this works.

Since

$$(1+ap)(1+bp) = 1+ap+bp+abp^2 \equiv 1+(a+b)p \pmod{p^2},$$

there is an isomorphism θ from the multiplicative subgroup

$$\Gamma = \{1+ap : 0 \le a \le p-1\}$$

of $\mathbb{Z}_{p^2}^*$ to the additive group $\mathbb{Z}_p$ defined by $\theta : 1+ap \mapsto a$. Equivalently, for $x \in \Gamma$, we have $\theta(x) = (x-1)/p$. Because $\mathbb{Z}_{p^2}^*$ has order $p(p-1)$ and is abelian, it has a unique subgroup of order p. Since Γ has order p, it contains all the order p elements of $\mathbb{Z}_{p^2}^*$. In particular, $g_p \in \Gamma$. Now, because θ is an isomorphism and $g_p \in \Gamma$, we have that $\theta(g_p^t) = t\theta(g_p)$ for all t. Thus discrete logarithms to the base g_p in the group Γ can be efficiently found provided that p is known.

Let (c,d) be a ciphertext generated by EPOC-2 encryption applied to a message m. Then $c \equiv g^r h^e \pmod{p^2}$ since the congruence holds modulo n. Therefore

$$c^{p-1} \equiv g^{r(p-1)} h^{e(p-1)} \equiv g_p^r \pmod{p^2}$$

because $g_p = g^{p-1} \bmod p^2$ and $h^{(p-1)} \equiv h_0^{p^2 q(p-1)} \equiv 1 \pmod{p^2}$. Applying the isomorphism θ gives $\theta(c^{p-1}) = \theta(g_p^r) = r\theta(g_p)$. Hence $r = \theta(c^{p-1})/\theta(g_p) \bmod p$.

Now that r is recovered, the EPOC-2 decryption algorithm correctly recovers the plaintext m by deriving the symmetric cipher key from r in exactly the same way as done for the EPOC-2 encryption.

It just remains to check if the ciphertext (c,d) passes step 6 of EPOC-2 decryption. If (c,d) was generated correctly, then $c \equiv g^r h^e \pmod{n}$. Since q divides n, it follows that $c \equiv g^r h^e \pmod{q}$. Thus step 6 will be passed.

4 Security of EPOC-2

The security objective of any public-key encryption scheme is *semantic security against adaptive chosen-ciphertext attacks* [7, 23]. We denote this notion of security by IND-CCA (indistinguishability against adaptive chosen-ciphertext attacks). Informally, this means that an adversary can learn nothing about the plaintext corresponding to a given ciphertext c, even when the adversary is allowed to obtain the plaintexts corresponding to ciphertexts (not equal to c) of its choice.

Another equivalent formulation of IND-CCA is the following. The goal of an adversary who launches an attack against a legitimate entity is to find two plaintexts for which the adversary is able to distinguish whether a challenge ciphertext is the encryption of the first plaintext or the second plaintext. To achieve this goal, the adversary is permitted to access a decryption oracle that will decrypt any ciphertext of the adversary's choosing, with the exception of the challenge ciphertext. The encryption scheme is IND-CCA if no such adversary exists.

EPOC-2 bases its security on the Okamoto-Uchiyama Integer Factorization (OUIF) problem, defined as follows. Let $n = p^2q$, where p and q are primes of the same bitlength. The *OUIF problem* is to find p, given n. The *OUIF assumption* is that there is no efficient algorithm for solving the OUIF problem.

4.1 Sketch of the Security Proof

The EPOC-2 submission [20] includes a proof that the EPOC-2 encryption scheme is IND-CCA in the random oracle model under the OUIF assumption. In the random oracle model, the hash function employed is modeled by a random function.

The proof of basic security (message recovery under passive attacks) of the Okamoto-Uchiyama (OU) encryption scheme [21] is similar to the proof of basic security for the Rabin-Williams encryption scheme. The security proof of the Fujisaki-Okamoto (FO) encryption padding scheme [6] is similar to (but simpler than) Bellare and Rogaway's security proof for the Optimal Asymmetric Encryption Padding (OEAP) scheme [2]. But there are some improvements of the FO proof over the OAEP proof. Together, the security proofs of the OU and FO schemes give a security proof of the EPOC-2 encryption scheme.

Suppose that A is a successful chosen-ciphertext distinguisher for EPOC-2. We would like to show that no such A exists under the conditions that the OUIF problem is intractable and the functions K and E are random oracles. To accomplish this, we convert such an A into a solver of the OUIF problem.

Consider an adversary A that only launches a chosen-plaintext attack, not an adaptive chosen-ciphertext attack. This is a weaker adversary, and therefore a proof of its non-existence should be easier.

This adversary A needs to find two equal-length messages m_0, m_1 such that A can subsequently guess b given (c, d), a legitimately generated EPOC-2 encryption of m_b, where b is selected at random from $\{0, 1\}$. Since $d = S_{K(r,P)}(m_b)$ and K is a random oracle, clearly A must not be able to determine b from $S_t(m_b)$ where t is random. This requirement clearly holds in the case where S is the one-time pad. In the case of all other symmetric ciphers S, this requires some security property from S. Assuming that S possesses this security property, then d alone is insufficient for A to determine b. Now $c = g^r h^e \bmod n$, where e is random, being the output of a random oracle. More precisely $e = E(m_b \| r \| d)$ (ignoring the parameter P). Using the following trick, we can ensure that A has no chance of learning b unless A is able to compute r explicitly—when generating (c,d), the simulator (implicitly) generates e at random, without actually selecting b. Certainly A cannot guess a value that has not yet been determined.

The only way to force a value of b to be associated with (c,d) is for the adversary to query the random oracle E with $(m_b \| r \| d)$. If A makes such a query, it has taken an element $z = g^r h^e \bmod n$ and extracted the value of r. Therefore A is effectively a kind of discrete logarithm problem solver. Now we can convert this A into an OUIF problem solver as follows.

On input n, choose $g, h_0 \in_R [0, n-1]$ and compute $h = h_0^n \bmod n$. It is the case that $\gcd(g,n) = 1$, the order of $g^{p-1} \bmod p^2$ is p, and $\gcd(h,n) = 1$ with overwhelming probability. Now run A with input n. After A has selected m_0 and m_1, choose $u \in_R [0, n-1]$ and $d \in_R \{0,1\}^{|m_0|}$, compute $c = g^u \bmod n$, and return (c,d) as the encryption of m_b (with b still unspecified). For each query of the form $(m_b \| r \| d)$ that A makes to the E-oracle, compute $t = \gcd(u - r, n)$. For the “right” query, i.e., the one for which $g^r h^e \equiv c \pmod{n}$ with $e = E(m_b \| r \| d)$, we have

$$g^u \equiv g^r h^e \pmod{n}.$$

Raising both sides to the power $p-1$ gives

$$g^{(p-1)u} \equiv g^{(p-1)r} h_0^{(p-1)p^2 q e} \pmod{n}.$$

Since $h_0^{(p-1)p} \equiv 1 \pmod{p^2}$, this yields

$$g_p^u \equiv g_p^r \pmod{p^2},$$

and thus $u \equiv r \pmod{p}$. With overwhelming probability, we have $u \neq r$, in which case f is a non-trivial factor of n.

4.2 Random Oracle Assumption

As with the known security proofs of RSA-OAEP, the security proof for EPOC-2 takes place in the random oracle model, and therefore does not imply security in the real world where the hash function is no longer a random function. This perspective was given some credence by Canetti, Goldreich and Halevi [5] who provided examples of signature and encryption schemes which they proved secure in the random oracle model, but which are insecure with *any* reasonable instantiation of the random function. However, the encryption and signature schemes in [5] are rather contrived, and a security proof in the random oracle model is now widely accepted as providing valuable *heuristic* evidence for security. Additionally, the security proof of EPOC-2 implies that any attack on EPOC-2 must either solve the OUIF problem, or exploit some properties of the hash function employed.

However, it is not clear what security properties EPOC-2 requires from the actual hash functions used. Clearly, if the key derivation function K were a constant, then there is no security because the symmetric cipher key would be constant. Once this constant key is known, then anyone can decrypt EPOC-2 ciphertext. Thus, at the very least, a non-constant K is required for EPOC-2. Certainly there are more requirements for K, but it is not at all clear what these requirements would be. Similar observations apply to the mask generation function E.

4.3 Tightness of the Reduction

Security proofs typically work by providing a *reduction* from the solution of a problem P to the breaking of a cryptographic scheme S. That is, one shows how P can be solved if one is given an oracle for breaking S. In the case of the EPOC-2 proof, S is EPOC-2 while P is the OUIF problem. The success of the reduction is measured by the tightness of the reduction. More precisely, a reduction is *tight* if when an adversary can break S in time t with probability ε, then the reduction algorithm that solves P runs in time t' with success probability ε', where $t' \approx t$ and $\varepsilon' \approx \varepsilon$. A tight reduction is important because one then has the assurance that the security level of S is very closely related to the security level of P. Thus one can select security parameters in S to be of the same size as the parameters that make P intractable against all known attacks. For example, if the reduction in a security proof of EPOC-2 is tight, then one can use 1152-bit moduli n and be assured that breaking EPOC-2 will require roughly the same amount of work as it takes to solve the OUIF problem for 1152-bit moduli.

The security proof of EPOC-2 has a very tight reduction (see Theorem 3.1 of [20]). Thus the security proof of EPOC-2 provides far greater assurances than the security proof of RSA-OAEP whose (known) security proofs are not tight enough to provide any meaningful assurances. This is one major advantage of EPOC-2 over RSA-OAEP.

4.4 Integer Factorization Problem

Recall that EPOC-2 bases its security on the assumption that factoring integers n of the form p^2q, where p and q are primes of the same bitlength, is intractable. This section provides a brief summary of the state-of-the-art in algorithms for the integer factorization problem.

PROBLEM DEFINITION. The *general integer factorization problem* is the following: given an integer n, determine the prime factorization of n.

TRIAL DIVISION. Once it is established that an integer is composite, before expending vast amount of time with more powerful techniques, the first thing that should be attempted is trial division by all "small" primes. Here "small" is determined as a function of the size of n. As an extreme case, trial division can be attempted by all primes up to $\sqrt{n}$. If this is done, trial division will completely factor n but the procedure will take roughly $\sqrt{n}$ divisions in the worst case when n is a product of two primes of the same size.

POLLARD'S RHO FACTORING ALGORITHM. Pollard's rho algorithm is a special-purpose factoring algorithm for finding small factors of a composite integer. Let $f : S \to S$ be a random function, where S is a finite set of cardinality n. Let x_0 be a random element of S, and consider the sequence $x_0, x_1, \dots$ defined by $x_{i+1} = f(x_i)$ for $i \geq 0$. Since S is finite, the sequence must eventually cycle, and consists of a tail of expected length $\sqrt{n\pi/8}$ followed by an endless repeating cycle of expected length $\sqrt{n\pi/8}$. A problem that arises in the factorization problem is of finding distinct indices i and j such that $x_i = x_j$. The expected time for finding such a collision is $O(\sqrt{p})$ where p is a prime factor of n.

POLLARD'S P-1 FACTORING ALGORITHM. Pollard's $p-1$ factoring algorithm is a special-purpose factoring algorithm that can be used to efficiently find any prime factors p of a composite integer n for which $p-1$ is smooth with respect to some relatively small B. Let n be an integer having a prime factor p such that $p-1$ is B-smooth. The running time of Pollard's $p-1$ algorithm for finding the factor p is $O(B \ln n / \ln B)$ modular multiplications.

ELLIPTIC CURVE FACTORING. One of the most powerful special-purpose factoring algorithms is the *elliptic curve factoring method (ECM)* which was invented by H. Lenstra in 1985 [13]. Its running time depends on the size of the prime factors of n, so the algorithm tends to find small factors first. The largest prime factor found thus far by ECM is a 55-decimal digit (187-bit) prime factor of a 112-decimal digit (371-bit) number reported by Izumi Miyamoto in October 2001. If both the prime factors of an EPOC-2 modulus n are at least 256 bits in length, then it is extremely unlikely that the ECM will be able to factor n. Thus, the ECM does not have a significant impact on the security of EPOC-2.

QUADRATIC SIEVE FACTORING. The quadratic sieve factoring algorithm has the same expected running time as the elliptic curve factoring algorithm in the special case when n is the product of two primes of equal size. However, for such numbers, the quadratic sieve is superior in practice.

NUMBER FIELD SIEVE FACTORING. The number field sieve (NFS) is the most efficient general-purpose factoring algorithm known. It has two main parts: the sieving stage and the matrix stage. The sieving stage, where most of the work is done, can be efficiently parallelized on a network of workstations. In contrast, existing techniques to solve the matrix stage seem to work well only when done on a single large parallel computer. The expected running time of the NFS for factoring an integer n is

$$O(\exp((1.923 + o(1))(\log n)^{1/3}(\log\log n)^{2/3})).$$

FACTORING $n = p^2q$. Although it is not known whether $n = p^2q$ is easier to factor than $n = pq$, some special algorithms to factor $n = p^2q$ have been studied in [1, 22]. These techniques are specific to the elliptic curve factoring method, and are at best a few times faster than the traditional ECM. The fastest algorithm known for factoring both integers n of the form $n = p^2q$ and $n = pq$ (where the primes p and q have the same bit length) remains the number field sieve method, whose running time depends only on the bit length of n.

FACTORING $n = p^rq$. Boneh, Durfee and Howgrave-Graham [4] presented an algorithm for factoring $n = p^rq$ with large r using the LLL algorithm. Their algorithm, however, is only effective for the case where r is large (at least $(\log p)^{1/2}$). If r is constant or small, the running time of their algorithm is exponential in the bitsize of n. For $n = p^2q$ (where $r = 2$), their algorithm is less efficient than the ECM and NFS methods.

SUMMARY. If n is carefully chosen, namely $n = p^2q$ where p and q are randomly chosen primes of the same bitlength which is at least 384 (so n has bit length at least 1152), then all known algorithms for factoring n are thwarted.

4.5 Parameter Recommendations

RECOMMENDED PARAMETERS. The EPOC-2 specification sets the following minimum security requirements:

- $k \geq 342$ (so the bitlength of n is at least 1024).
- $v \geq 32$ (so the bitlength of e is at least 716).

In addition, the following domain parameters are recommended:

- $k = 384$ (so the bitlength of n is 1152).
- Hash function = SHA-1.
- K = KDF2(SHA-1).
- E = MGF1(SHA-1).
- S = Camellia in CBC mode with IV=0.
- Key length of S = 128 bits.
- v = 192 (so the bitlength of e is 960).

INCOMPATIBLE SECURITY LEVELS. The security level provided by Camellia with a 128-bit key is believed to be 128 bits. However, a 1152-bit modulus n only provides about 80 bits of security against factoring by the NFS algorithm. Thus, if 128 bits of security is really desired, then a larger value for k should be recommended. (However, if only 80 bits of security is required, then the recommended parameters are satisfactory.)

SELECTING PARAMETERS FOR LONG-TERM SECURITY. Lenstra and Verheul [12] performed an extensive and careful study of the key sizes for symmetric-key encryption schemes, RSA, discrete logarithm systems, and elliptic curve systems. Their study incorporates both software and hardware attacks, and takes into account the continual improvements in hardware, and hardware costs. Assuming that the NFS remains the best algorithm for integer factorization, they estimate that 1024-bit RSA will provide the same level of security in the year 2002 as the Data Encryption Standard (DES) provided in the year 1982. (DES was considered very secure in 1982 for banking applications.) Similarly, 2048-bit RSA will provide the same level of security in the year 2023 as DES provided in the year 1982, and 3072-bit RSA will provide the same level of security in the year 2038 as DES provided in the year 1982.

In a recent study, Lenstra [11] provides estimates of RSA key lengths required to provide equivalent levels of security as the common symmetric-key encryption schemes DES, 2K3DES (two key Triple-DES), 3K3DES (three key Triple-DES), AES-128 (128-bit Advanced Encryption Standard), AES-192 and AES-256 [18]. Lenstra's estimates are reproduced in Table 1. The entries of the table are to be interpreted as follows. The entry of 3296 for AES-128 in the year 2020 means that one should used a 3296-bit RSA modulus to protect a 128-bit symmetric key if in the year 2020 one desires the same amount of resistance to the NFS algorithm for factoring RSA moduli as exhaustive key search on AES-128.

Year	DES	2K3DES	3K3DES	AES-128	AES-192	AES-256
2001	416	1333	1941	2644	6897	13840
2010	518	1532	2189	2942	7426	14645
2020	647	1773	2487	3296	8042	15574
2030	793	2035	2807	3675	8689	16538

Table 1: Matching RSA modulus sizes.

Assuming that factoring RSA moduli (of the form $n = pq$) remains equally difficult as factoring EPOC-2 moduli (of the form $n = p^2q$), Lenstra and Verheul's estimates for RSA security apply equally well to EPOC-2 security. If EPOC-2 is to be used for long-term security, it is imperative that the EPOC-2 specification provide guidance on the choice of domain parameters (e.g., k, v, hash function, symmetric cipher) corresponding to increasing levels of security.

4.6 Implementation Attacks

The security proof of EPOC-2 does not take into account side-channel attacks such as power analysis [10], timing attacks [9], differential fault analysis attacks [3], and attacks which exploit weak random or pseudorandom number generators [8].

Consider the following attack, modeled on the attack of Manger [14] on RSA-OAEP. An adversary randomly selects $c \in [0, n-1]$ and d, and sends (c,d) to a decryptor. During EPOC-2 decryption, the decryptor must verify the following two conditions: (I) the range of the value r recovered (step 3 of Section 3.4); and (II) the correct re-encryption of c modulo q (step 6 of Section 3.4). It so happens that Test I is performed first in the EPOC-2 specification [19]. In the attack there is roughly a $\frac{1}{2}$ chance that r will pass (the actual probability depends on the difference between p and 2^k). So, even with random (c,d), the recovered r will be in the correct range, in which case the decryptor performs Test II.

Suppose that an implementation of EPOC-2 is designed such that when Test I fails, then Test II is skipped. This implementation is especially vulnerable to timing and power analysis attacks since Test II consumes considerable time and power. The consumption or non-consumption of time and power resources will indicate whether Test I passed or failed. Furthermore, since Test I passes or fails with roughly equal frequency, every such analysis is likely to reveal approximately a bit of information about the value of r. In particular, the attacker will be able to test random values of c and determine whether the r values they generate lies in a certain interval.

This side-channel leakage of information *might* be exploitable. For example, the frequency of passing and failing reveals something about the value of p. Moreover, the attacker *may* be able to leverage the fact that the map from c to r is a *homomorphism*: if $c_1 \mapsto r_1$ and $c_2 \mapsto r_2$, then $c_1c_2 \mapsto (r_1 + r_2) \bmod p$.

We do not know how to turn these observatoins into an actual attack on EPOC-2. However, to mitigate against the possibility of such attacks, a cautious implementor would ensure that:

1. Test II is always performed;
2. Test II is performed before Test I; and
3. Test I is skipped if Test II fails.

The first recommendation is to avoid the potential timing and power analysis attacks mentioned above. The other two recommendations are to avoid any risk of leakage arising from Test I without first checking strong validity with Test II.

Certainly more work needs to be done on analyzing the resistance of EPOC-2 to implementation attacks.

5 Performance

5.1 Encryption and Decryption Operations

ENCRYPTION. The computationally expensive operations in EPOC-2 encryption are the two modular exponentiations $g^r \bmod n$ and $h^e \bmod n$. Recall that the computationally expensive step in RSA encryption is the modular exponentiation $m^e \bmod n$. EPOC-2 encryption is considerably slower than RSA encryption when a small encryption exponent (such as $e = 3$ or $e = 2^{16} + 1$) is used in the latter. EPOC-2 encryption should be a little faster than RSA encryption if arbitrary exponents are used for RSA since the EPOC-2 exponents r and e are roughly one-third and two-thirds, respectively, the bitsize of an arbitrary exponent, and the exponentiations in EPOC-2 can be sped up using Shamir's trick for double exponentiation.

DECRYPTION. In [20] it is claimed without adequate justification that EPOC-2 decryption is significantly faster than RSA decryption. We therefore include a brief analysis of how EPOC-2 decryption compares to RSA decryption.

We make the following assumptions in our analysis: (i) multiplication of two arbitrary numbers modulo an m-bit modulus costs m^2 bit operations; and (ii) exponentiation by an l-bit exponent takes on average $3l/2$ modular multiplications.

EPOC-2 decryption involves the following exponentiations: (i) one exponentiation by $p-1$ modulo p^2 for a cost of $(3k/2)(2k)^2$ (bit operations); and (ii) two exponentiations modulo q for a cost of $2(3k/2)k^2$. The total cost is $9.0k^3$. With Shamir's trick, the latter two exponentiations can be sped up to $1.75k^3$ thereby lowering our estimate to $7.75k^3$. To normalize our comparison, let $K = 3k$ be the bitsize of the public modulus n. The cost is now $(7.75/27)K^3 \approx 0.29K^3$.

Naive RSA decryption with a modulus of the same bitsize K costs $(3K/2)K^2 = 1.5K^3$, which is considerably more than EPOC-2. But CRT-based RSA decryption costs $2((3/2)(K/2))(K/2)^2 = 0.375K^3$, which seems very close to our estimate for EPOC-2.

In the EPOC-2 self-evaluation document [20], the following timings are presented for the efficiency of RSA-OAEP and EPOC-2 encryption and decryption. The moduli n for both EPOC-2 and RSA-OAEP have bitlength 1152. An encryption exponent $e = 2^{16} + 1$ is used for RSA.

EPOC-encryption:	1472
EPOC-2 decryption:	192
RSA-OAEP encryption:	17
RSA-OAEP decryption:	432

We would expect RSA encryption to take 17 modular multiplications, and RSA decryption (with CRT) to take the equivalent of $0.375 \times 1152 = 144$ modular multiplications. Thus we expect RSA decryption to be about 8 times slower than RSA encryption. However, in the above table, RSA decryption is 25 times slower than RSA encryption. The document [20] does not mention whether these performance figures are theoretical estimates or timings obtained from an actual implementation. Therefore, it is impossible for us to judge the validity of the performance numbers from [20]. Because of the closeness of the costs in our rather crude analysis, a fairer comparison would require full implementations of both RSA-OAEP and EPOC-2 before definite conclusions can be drawn.

DISCUSSION. In many applications, fast decryption is critical. For example, an SSL server has to decrypt messages from a multitude of different clients. Frequent decryption proves a significant burden to the server, and efficient decryption would help to reduce costs. In some applications, decryption can only be performed on a device that can securely store private keys, such as a smart cards—often these devices have very limited performance capabilities. EPOC-2 does not appear to offer significant savings on decryption over RSA-OAEP.

In some applications, fast encryption is critical. For example, when sending a single message to a large number of recipients, rapid encryption is helpful. For such applications, RSA-OAEP with a small encryption exponent offers significant benefits over EPOC-2.

5.2 Using EPOC-2 in Security Protocols

Many security protocols, including SSL, TLS, WTLS, IPSec, and S/MIME, use public-key encryption only for key transport. The use of provably secure public-key encryption for key transport only ensures confidentiality of the symmetric key being transported, and not of the data subsequently encrypted with that key.

The reasons that security protocols use public-key encryption for key transport only include the following: (i) the existing standardized public-key encryption schemes such as RSA-PKCS1-v1.5 [24] and RSA-OAEP [25] did not allow for arbitrarily long plaintexts; and (ii) these public-key encryption schemes were not fast enough to directly encrypt the entire content.

However, EPOC-2 is a hybrid scheme that uses a symmetric cipher to encrypt the plaintexts. Thus, EPOC-2 supports plaintexts of arbitrary lengths, and EPOC-2 for direct content encryption is as fast (for long plaintexts) as using public-key encryption for key transport combined with a symmetric cipher for content encryption.

Therefore when deploying EPOC-2 there is good reason to use it for direct content encryption, not just for key transport. But to do this requires considerable care.

Consider the TLS protocol. Currently, TLS uses key transport to send a master key during a "handshake". This master key is used for content encryption and content authentication. The content is divided into records of at most 2^{14} bytes and is encrypted continuously, record-by-record. If a stream cipher is used, the stream is continued for each record—it is not reinitialized, since this would be insecure. If a block cipher is used in CBC mode, the last ciphertext block of the previous record is used as the IV for the next record. In effect, the master symmetric key is only used once. (Only when a new handshake or new key change message is sent, will a new master symmetric key be established and symmetric encryption be restarted with a new key.)

The same method could be applied with EPOC-2: an entire series of TLS records could be encrypted in a single EPOC-2 encryption simply by continuing the symmetric cipher phase of EPOC-2. But this does not address the issues of message integrity and authentication. Because the current specification of EPOC-2 does not support single-pass processing (see §5.3), the above method would be very impractical since the TLS client and server would have to buffer many records in order to complete the final phase of EPOC-2 encryption (but see §5.3 for how to modify EPOC-2 to support single-pass processing). Furthermore, the TLS protocol authenticates each record. This allows a tampered record to be rejected before it is used. However, EPOC-2 only checks integrity after completely processing the message. Thus the use of EPOC-2 in this mode (even modified to allow single-pass processing) would have the disadvantage that records could not be checked for integrity until a new handshake or a new key was otherwise established. Of course, EPOC-2 could be used for key transport only, but then its provable security advantages may be lost.

5.3 Stream Processing

As noted by Shoup [26], the EPOC-2 encryption and decryption procedures do not support single-pass or stream processing. To illustrate what this means, why EPOC-2 cannot do it, and why this may be problematic, consider the following scenarios.

1. Suppose that we have a 20 Gbyte hard-disk, with 11 Gbytes utilized. Now suppose that we wish to encrypt that data on the hard disk. According to EPOC-2 encryption, we would take the 11 Gbytes of data to be the message m, and encrypt using symmetric cipher S and key $K(r,P)$ to get a value d. Certainly, d would be about 11 Gbytes. Now d and m cannot both fit on the hard drive, because there is only space for 20 Gbytes. To get around this, we can overwrite the plaintext data m with the data d incrementally during the symmetric enciphering process. Once this is done, however, $E(m \| r \| d \| P)$ cannot be computed. One solution is to compute $E(m \| \ldots)$ as m is being enciphered—this is possible since most hash functions (such as SHA-1) process data from beginning to end. But then we must process d, which requires a whole *second pass* over the 11 Gbytes of data, which could be slow.

   EPOC-2 decryption is more difficult in this scenario. Here we would recover r and begin to decipher d. This would make m available incrementally and thereby allow the computation $E(m \| \ldots)$ without making an additional pass. But because there is not enough space on the hard disk, d would have be discarded during its deciphering. As each segment of d is decrypted it is discarded and replaced by the corresponding decrypted segment of m. But during EPOC-2 decryption, the computation of e becomes problematic because d has been erased. One way to get around this is to do a second pass of encrypting-then-redecrypting to temporarily recover d. This second pass is even more costly than the second pass needed for encryption.

2. Suppose that we are working on a device with low RAM, such as a smart card, PDA or cell phone. Then to encrypt or decrypt long messages, the second pass problem may be infeasible, because it always requires buffering the plaintext or ciphertext.

3. Suppose that EPOC-2 is being used by some intermediate gateway server that decrypts messages, re-encrypts, and forwards them on. If the gateway was a channel for many messages from various parties on two networks, the buffering required to make second passes could easily overwhelm the memory of the gateway.

There are some solutions to allow one-pass processing with EPOC-2.

One solution is to use EPOC-2 for key transport only—encrypt a symmetric cipher key using EPOC-2 and then use single-pass processing with that symmetric key. This solution has some problems. First, the provable security advantage of EPOC-2 is perhaps lost. Indeed, most symmetric ciphers do not ensure message integrity, and therefore may permit malleability attacks and adaptive chosen-ciphertext attacks. Thus, considerable care and thought would have to go in the symmetric cipher method used, in particular, with some secure method for adding integrity.

A better solution is to modify EPOC-2 slightly. For example, $e = E(m \| r \| d \| P)$ could be changed to $e = E(m_1 \| d_1 \| m_2 \| d_2 \| \dots \| r \| P)$, where m_i and d_i are corresponding segments of m and d of convenient lengths. Most block ciphers and all stream ciphers, including the one-time pad, would easily allow for such a modification. This modification allows stream processing, because e can be calculated during the enciphering of m in the encryption procedure, or deciphering of d in the decryption procedure. There is no need to buffer data.

Although this modification solves the stream processing problem from a computational standpoint, it does not solve it from a communications standpoint. If the communications are streamed, then because c precedes d in the EPOC-2 ciphertext, d must be buffered by the sender until c is computed. If EPOC-2 is changed so that the format is (d, c), then the sender does not have this problem. But given (d, c), the recipient must buffer the entirety of d before decryption can begin, since c is needed in order to start deciphering d. It seems to be a fundamental feature of EPOC-2, no matter what format is used for the ciphertext, that either the originator or the recipient cannot use streamed communication. At least one side has to buffer data.

The modification suggested above does not permit the decrypted plaintext stream to be used in a stream fashion. This is because the plaintext stream generated is not checked for integrity until the very end. It is therefore not safe to use until the end because it could have been modified.

CONTINUOUS INTEGRITY. A better way to do stream processing is to use *segmentation* or *continuous integrity verification*. In this method, the ciphertext is divided into segments each having a certain amount of integrity that is immediately verifiable. In this approach the corresponding segments of plaintext are immediately usable. This segmented approach is already widely used in practice, for example in the IPSec, SSL, TLS and WTLS security protocols. Small segments of data are protected by the combination of message authenticate code (MAC) and a symmetric cipher.

The current version of EPOC-2 does not support stream ciphering so clearly does not support segmented stream ciphering. Moreover, it does not seem possible to modify EPOC-2 so as to allow for efficient segmented stream ciphering because EPOC-2 does not use a MAC to check for integrity, but a more costly number-theoretic approach to integrity.

6 Conclusions

EPOC-2 is an encryption scheme for encrypting arbitrarily long plaintext messages.

SECURITY. EPOC-2 has been proven to be semantically secure against adaptive chosen-message attacks in the random oracle model under the assumption that factoring integers of the form p^2q is intractable. One advantage of EPOC-2 over competing systems is that its security is based on integer factorization, unlike schemes such as RSA-OAEP whose security is based on the potentially easier problem of finding *e*th roots modulo a composite integer. Moreover, the reduction in the security proof is very tight, so one can be assured that the security level of EPOC-2 is closely related to the security level of the integer factorization problem.

The proof of security of EPOC-2 appears to be sound. However, more public scrutiny of the proof is recommended—proving the security of public-key encryption schemes can be extremely tricky and may have subtle flaws which cannot be discovered by casual examination of the proofs. It is unlikely that the security proof of EPOC-2 has received much public attention since it has not been published in its full form. In particular, the resistance of EPOC-2 to implementation attacks should be further studied.

The security proof provided for EPOC-2 uses the one-time pad as the symmetric cipher—however, the recommended symmetric cipher is the block cpiher Camellia. More guidance is needed on the desired properties of the symmetric cipher employed in an implementation of EPOC-2.

The EPOC-specification provides recommendations only for minimum security levels, and a set of domain parameters for one (fixed) level of security. Guidance should be provided on the choice and size of domain parameters for long-term security.

PERFORMANCE. One major disadvantage of EPOC-2 is that it is not well suited for stream processing of large plaintexts. This can be problematic in a variety of environments, e.g., if buffering large messages is not feasible.

COMPARISON WITH RSA-OAEP. EPOC-2 and RSA-OAEP can each claim comparative advantages and hence it is not possible to say with certainty that one scheme is better than the other. The following is a summary of the key differences.

1. RSA-OAEP encryption (with small encryption exponent) is significantly faster than EPOC-2 encryption. On the other hand, EPOC-2 decryption *may* be faster than RSA-OAEP, although more study is needed before this can be concluded with certainty.

2. Both EPOC-2 and RSA-OAEP are provably secure in the random oracle model. The underlying hard problem in EPOC-2 is integer factorization, while the underlying hard problem in RSA-OAEP is the problem of finding *e*th roots modulo *n* which is potentially easier than factoring. Furthermore, the reduction in the security proof of EPOC-2 is much tighter than the reduction for RSA-OAEP. Thus, the assurances provided for the security of EPOC-2 in practice are much higher than for RSA-OAEP.

3. RSA-OAEP has been more widely studied than EPOC-2. In particular, more work has been published on the efficient implementation of RSA, and on the resistance of RSA to various implementation attacks.

4. EPOC-2 is not well suited to streaming, while RSA-OAEP does not have this drawback.

References

[1] L. Adleman and K. McCurley, “Open problems in number theoretic complexity II”, *Algorithmic Number Theory–Proceedings of ANTS-I*, Lecture Notes in Computer Science, **877** (1995), Springer-Verlag, 291-322.

[2] M. Bellare and P. Rogaway, “Optimal asymmetric encryption”, *Advances in Cryptology–Eurocrypt ’94*, Lecture Notes in Computer Science, **950** (1995), Springer-Verlag, 92-111. Full version available at http://www.cs.ucdavis.edu/~rogaway/papers/index.html

[3] D. Boneh, R. DeMillo and R. Lipton, “On the importance of checking cryptographic protocols for faults”, *Advances in Cryptology–Eurocrypt ’97*, Lecture Notes in Computer Science, **1233** (1997), Springer-Verlag, 37-51.

[4] D. Boneh. G. Durfee and N. Howgrave-Graham, “Factoring $N = p^r q$ for large r”, *Advances in Cryptology–Crypto ’99*, Lecture Notes in Computer Science, **1666** (1999), Springer-Verlag, 326-337.

[5] R. Canetti, O. Goldreich and S. Halevi, “The random oracle methodology, revisited”, *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing*, 1998, 209-218.

[6] E. Fujisaki and T. Okamoto, “Secure integration of asymmetric and symmetric encryption schemes”, *Advances in Cryptology–Crypto ’99*, Lecture Notes in Computer Science, **1666** (1999), Springer-Verlag, 537-554.

[7] S. Goldwasser and S. Micali, “Probabilistic encryption”, *Journal of Computer and System Sciences*, **29** (1984), 270-299.

[8] J. Kelsey, B. Schneier, D. Wagner and C. Hall, “Cryptanalytic attacks on pseudorandom number generators”, *Fast Software Encryption–FSE ’98*, Lecture Notes in Computer Science, **1372** (1998), Springer-Verlag, 168-188.

[9] P. Kocher, “Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems”, *Advances in Cryptology–Crypto ’96*, Lecture Notes in Computer Science, **1109** (1996), Springer-Verlag, 104-113.

[10] P. Kocher, J. Jaffe and B. Jun, “Differential power analysis”, *Advances in Cryptology–Crypto ’99*, Lecture Notes in Computer Science, **1666** (1999), Springer-Verlag, 388-397.

[11] A. Lenstra, “Unbelievable security: Matching AES security using public key systems”, *Advances in Cryptology–Asiacrypt 2001*, Lecture Notes in Computer Science, **2248** (2001), Springer-Verlag, 67-86

[12] A. Lenstra and E. Verheul, “Selecting cryptographic key sizes”, *Public Key Cryptography–Proceedings of PKC 2000*, Lecture Notes in Computer Science, **1751** (2000), Springer-Verlag, 446-465. Full version to appear in *Journal of Cryptology*.

[13] H. Lenstra, “Factoring integers with elliptic curves”, *Annals of Mathematics*, **126** (1987), 649-673.

[14] J. Manger, “A chosen ciphertext attack on RSA optimal asymmetric encryption padding (OAEP) as standardized in PKCS#1 v2.0”, *Advances in Cryptology–Crypto 2001*, Lecture Notes in Computer Science, **2139** (2001), Springer-Verlag, 230-238.

[15] A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[16] National Institute of Standards and Technology, *Secure Hash Standard (SHS)*, FIPS Publication 180-1, 1995.

[17] National Institute of Standards and Technology, *Secure Hash Standard (SHS)*, Draft FIPS 180-2, 2001. Available from http://csrc.nist.gov/encryption/tkhash.html

[18] National Institute of Standards and Technology, *Advanced Encryption Standard (AES)* FIPS Publication 197, 2001.

[19] NTT Information Sharing Platform Laboratories, “EPOC-2 specification”, September 26 2001.

[20] NTT Labs, “Self evaluation of EPOC-2”, 2001.

[21] T. Okamoto and S. Uchiyama, “A new public-key cryptosystem as secure as factoring”, *Advances in Cryptology–Eurocrypt ’98*, Lecture Notes in Computer Science, **1403** (1998), 308-318.

[22] R. Peralta and E. Okamoto, “Faster factoring of integers of a special form”, *IEICE Trans. Fundamentals, E79-A*, **4** (1996), 489-493.

[23] C. Rackoff and D. Simon, “Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack”, *Advances in Cryptology–Crypto ’91*, Lecture Notes in Computer Science, **576** (1992), 433-444.

[24] RSA Laboratories, *PKCS#1 v1.5: RSA Encryption Standard*, November 1993.

[25] RSA Laboratories, *PKCS#1 v2.1: RSA Cryptography Standard* (Draft 2), January 5 2001. Available at http://www.rsa.com/rsalabs/pkcs/pkcs-1/

[26] V. Shoup, “A proposal for an ISO standard for public key encryption (version 2.0)”, September 17 2001. Available at http://shoup.net/papers/