A Weak-Randomizer Attack on RSA-OAEP with $e = 3$

Daniel R. L. Brown[*]

June 22, 2005

Abstract

Coppersmith's heuristic algorithm for finding small roots of bivariate modular equations can be applied against low-exponent RSA-OAEP if its randomizer is weak. An adversary that knows the randomizer can recover the entire plaintext message, provided it is short enough for Coppersmith's algorithm to work. In practice, messages are symmetric cipher keys and these are potentially short enough for certain sets of key sizes. Weak randomizers could arise in constrained smart cards or in kleptographic implementations. Because RSA's major use is transporting symmetric keys, this attack is a potential concern. In this respect, OAEP's design is more fragile than necessary, because a secure randomizer is critical to prevent a total loss of secrecy, not just a loss of semantic security or chosen-ciphertext security. Countermeasures and more robust designs that have little extra performance cost are proposed and discussed.

1 Introduction

Evidence exists that RSA encryption when the public exponent $e = 3$ is weaker than the general problem of inverting the raw RSA function with $e = 3$ at random values. Examples are Coppersmith's [Cop96] and Hastad's broadcast attack. (Boneh [Bon99] surveys some of these results.) Security proofs aim to rule out unforeseen attacks by reducing security to a hard problem. The RSA-OAEP encryption scheme [BR94] has a proof that reduces its security to the problem of inverting the RSA function. The proof applies even if $e = 3$, so therefore RSA-OAEP avoids the attacks mentioned above. Security proofs, however, rely on assumptions, and assumptions can fail. The RSA-OAEP encryption scheme is randomized, as necessary for semantic security and chosen-ciphertext security. In this paper, we examine what happens if the randomization is weak.

Our main observation is that when the randomizer is weak, then Coppersmith's algorithm for small roots of bivariate modular equations potentially leads to recovery of the RSA-OAEP plaintext if $e = 3$. This is more loss than one immediately expects from weak randomization, because one only expects semantic security or chosen-ciphertext security to be defeated. In this attack, a total loss of secrecy results. When RSA-OAEP is used to transport a strong cryptographic key, this could be disastrous (whereas semantic security or chosen-ciphertext attacks barely impact key transport.) The full analysis of this attack depends on the efficiency of Coppersmith's algorithm. The existing analysis is heuristic, but experiments indicate the algorithm is effective. Ongoing research will be needed to analyze the degree of vulnerability of OAEP to this attack.

[*]Certicom Research

2 Weak Randomizers

Weak randomizers could arise in various ways.

- A faulty implementation may use an insecure deterministic random number generator algorithm, or it may fail to supply sufficient entropy to a secure deterministic random number generator.
- A constrained system, such as a smart card, may not have sufficient entropy to generate random numbers securely. In a highly modular system, the message may arrive from a different source than the RSA-OAEP encryption component, so it cannot be assumed that if the message has sufficient entropy then the RSA-OAEP encryption component is also capable of supplying sufficient entropy.
- A kleptographic implementation may generate the secret randomizer according to a sequence known to an external adversary who can then decipher the RSA-OAEP ciphertexts. Unlike traditional kleptographic attacks, this attack is on the public key operations, not the private key generation algorithm. Public key operation implementations are often downloaded software, with no little or no authentication, as part of web browsers and email clients, whereas private key operations are more often implemented with secured hardware.

3 The Attack

In an OAEP-encoding of a message m, the number of bits that depend on the message is about $\ell(m)+\ell(h)$, where h is the hash function, and $\ell(\cdot)$ is the bit-length. All other bits are a function of the secret randomizer and some constant values. More precisely, the ciphertext may be expressed as

$$c = (x2^f + k + y)^e \mod n \tag{1}$$

where x and y are unknown integers of bit-lengths $\ell(h)$ and $\ell(m)$, respectively. The value k is known, being the exclusive-or of a constant and the MGF output of the (known) secret randomizer. This is a special case of:

$$c = f(x, y) \mod n \tag{2}$$

where f is a polynomial. Coppersmith's heuristic algorithm for finding small roots of modular bivariate polynomial equations may be applied to find x and y if they are small enough. Finding x and y of course gives the message m.

It remains to examine how small x and y need to be so that Coppersmith's algorithm can find them. A rough estimate is that x and y should have total bit-length no more than $1/e$ of the bit-length of n.

Table 1 reveals that fairly large messages can be recovered if the randomizer is weak. For example with a 3072 bit RSA key and exponent $e = 3$, then a 750-bit message can be recovered from a compromised randomizer. A 750-bit message could contain about 6 AES keys of 128 bits each. More to the point, in the SSL and TLS protocols, the message size is fixed to 48 bytes, or 384 bits. The attack is then applicable against 2048 bit RSA public keys used in the SSL and TLS

Bit Length			Approximate
RSA	Hash	Message	Ratio
1024	160	160	1/3
2048	224	440	1/3
3072	256	750	1/3
8192	384	2300	1/3
15360	512	4500	1/3

Table 1: Vulnerable Message Sizes for Various Key Sizes

protocols. Arguably, however, in a SSL/TLS implementation, because the message is a random value, it may be generated from the same source as the OAEP randomizer, and if that source is weak, then so is the message itself, meaning little is gained by recovering it. Most SSL/TLS implementation may thus be unaffected, but there may still be some that are vulnerable for various reasons. For example, the OAEP encryption may be implemented by a separate module, perhaps a hardware module, with an independent random number generator, or else the implementer did not realize the importance of the OAEP randomizer and simply used some weak random number generator, such as a hash of the system clock.

4 Countermeasures

Obviously, implementations should generate the secret randomizer with a secure random number generator, whenever possible. In certain circumstances this may not be feasible, however, and alternative countermeasures may be needed, such as the following:

- Deriving the secret randomizer as a function of both a weak random number generator and the plaintext message. Therefore, if the plaintext message has enough entropy to make it unguessable, then the secret randomizer will have enough entropy to make it unguessable too, thereby thwarting this attack. This is especially useful for key transport.

- Adding extra iterations to the Feistel ladder construct used in OAEP. This helps the entire input to the RSA function depend upon both the plaintext message and the secret randomizer. This helps if the plaintext message and the secret randomizer come from two separate sources of weak entropy whose total entropy is enough to prevent guessing, but whose individual entropies are low enough to permit guessing. For example, if the randomizer has 40 bits of entropy and the message also 40 bits of entropy, then the total would be 80, for a space of size 2^{80} which is considered infeasible to search today, and at least until 2010. By comparison a space of size 2^{40} is quite easily searched today.

- Using messages of the larger size. To be effective, however, one cannot just pad the message. If the padding is known to the attacker, then Coppersmith's algorithm can still be used, in just the same way as OAEP's padding is not an obstacle once known. Therefore, the extra length has to be unguessable in its own right.

5 Comments

Some comments about the weak randomizer attack follow.

5.1 Fragility of OAEP

A plausible misconception is that the design of OAEP scheme completely prevents unexpected problems, such as from application of Coppersmith's algorithm. To an extent this true because of OAEP has a security proof. More precisely, the security proof for OAEP shows that its security is as strong as the RSA problem for $e = 3$ is hard. Therefore, under the assumptions of the security proofs, even Coppersmith's root-finding algorithms cannot be used to learn information about OAEP-encrypted messages, unless it can also be used to solve the basic RSA problem for $e = 3$. More intuitively, the OAEP encoding eliminates all structure from the input to RSA function, leaving nothing left for Coppersmith's algorithm to use.

Why this paper's attacks works, despite the security proofs for OAEP, is that the assumptions of the proof are not completely realistic. In the context of this attack, the unrealistic assumption is a perfectly secret randomizer. (The security proofs also "assume" a random oracle model for a hash function, which is not completely realistic either. That is the philosophical issue, however, and not the issue for this paper.)

One way to view this kind of attack is that the design of OAEP is not as *robust* as possible, as explained below. In practice, components of any security system can fail. Any security system may suffer some security loss from a component failure. If the security loss is total, then that component may be defined as *critical.* A system that has a lot of critical components is defined as *fragile*, whereas a system that has few or no critical components is defined as *robust.* This paper shows how the randomizer in OAEP is a critical component, and therefore that OAEP is fragile in this respect.

Every public-key encryption scheme loses both its semantic and chosen-ciphertext security if its secret randomizer fails. However, this is not complete security loss, since the attacker does not automatically learn the whole plaintext or private key. For example, if a sufficiently random message is encrypted, then an adversary cannot recover the message, even if he knows the secret randomizer. Therefore, in general, the secret randomizer is not necessarily a critical component. In low-exponent RSA-OAEP, however, this paper has shown that the secret randomizer is critical, and therefore OAEP more fragile than a general public key encryption scheme needs to be.

5.2 A Variant Kleptographic Attack

We consider a variant kleptographic attack. Suppose that randomizer is secure, in the sense that the adversarial implementation cannot choose, modify or predict it, and has no side channel to reveal it. The adversarial implementation can, however, learn the secret randomizer. If adversarial implementation can then manipulate the plaintext message in such a way to arrange for the OAEP-encoded message to end in many trailing zeros. Now the ciphertext will be kleptographically vulnerable to an outsider. The attack of Coron, Joye, Naccache and Paillier's attack [CJNP00], can be applied to recover the full plaintext message. (See also §A for a description of a trivial case of Coppersmith's stereotyped message attack that can be applied here too.) Technically, this attack is detectable by the decryptor because the OAEP-encoded message ends in trailing zeros, so in some respects it is a limited attack from a kleptographic perspective.

5.3 Diffie-Hellman Based Key Establishment

In encryption schemes based on Diffie-Hellman key agreement, such as DHAES and ECIES, secrecy of the randomizer is inherent. In fact, in these systems, the randomizer is generally called the ephemeral private key. This terminology, *key*, has generally led to recognition that the randomizer requires full protection. Most implementations strive to protect the key. For low-exponent RSA-OAEP, it may be appropriate to call the randomizer a key.

Interestingly, the Menezes-Qu-Vanstone (MQV) key agreement scheme is slightly more robust in the event of weak randomizers. In MQV both sides have static private keys, which are presumably generated with strong random number generators. If the ephemeral keys become weak, then the certain security properties may be lost, such as forward secrecy, but the fundamental confidentiality of the agreed key is maintained.

Despite the above mitigating factors against weak randomizer attacks, Diffie-Hellman based schemes would also benefit from one of the alternative countermeasures suggested for OAEP, namely that of supplementing the weak randomizer with a function of the plaintext message.

6 Conclusion

The design of RSA-OAEP has been shown to be not as robust as possible. If the randomizer fails, then more security is lost than would be expected for a general public key encryption scheme. This applies mainly to very low RSA exponents, such as $e = 3$, and for messages of moderate length. Care should taken to avoid such problems with RSA-OAEP.

References

[Bon99] Dan Boneh, *Twenty years of attacks on the RSA cryptosystem*, Notices of the American Mathematical Society **46** (1999), no. 2, 203–213, `crypto.stanford.edu/~dabo/abstracts/RSAattack-survey.html`.

[BR94] Mihir Bellare and Phillip Rogaway, *Optimal asymmetric encryption*, Advances in Cryptology — EUROCRYPT '94 (Alfredo De Santis, ed.), LNCS, no. 950, IACR, Springer, May 1994, pp. 92–111.

[CJNP00] Jean-Sébastien Coron, Marc Joye, David Naccache, and Pascal Paillier, *New attacks on PKCS#1 v1.5 encryption*, Advances in Cryptology — EUROCRYPT 2000 (Bart Preneel, ed.), LNCS, no. 1807, IACR, Springer, May 2000, pp. 369–381.

[Cop96] Don Coppersmith, *Finding a small root of a univariate modular equation*, Advances in Cryptology — EUROCRYPT '96 (Ueli Maurer, ed.), LNCS, no. 1070, IACR, Springer, May 1996, pp. 155–165.

A Coppersmith's Stereotyped Message Attack on PKCS #1 v. 1.5

Weak randomizers also make PKCS #1 v. 1.5 insecure, if the message is short enough. Similar countermeasures about selection of the randomizer can be applied to PKCS #1 v. 1.5.

Another known attack on PKCS #1 v. 1.5 is the stereotyped message attack of Coppersmith. A special case of this attack is the message ends with a sufficient number of trailing zeroes. Then, the plaintext can easily be recovered just by computing a cube root, without even using Coppersmith's root-finding algorithms. Although this attack is known and quite trivial, its simplicity make it worth re-iterating here. Coron, Joye, Naccache and Paillier's [CJNP00] give another attack against PKCS #1 v. 1.5 for messages with trailing zeroes. Their attack requires two ciphertexts, not just one, and is more complicated than solving a cube root, but has the advantage that it needs fewer trailing zero bits.

Given a non-negative integer message m of a certain length and RSA public key (n, e), then the corresponding PKCS #1 v. 1.5 ciphertext is computed as

$$c \equiv (r2^f + m)^e \mod n, \tag{3}$$

where m has been converted to an integer, f is determined by the length of m, and r is a randomizer. In PKCS #1 v. 1.5, the randomizer is given by a random sequence of non-zero bytes.

Suppose that message m ends in g trailing bits, where $g \leq f$. Then $m = m'2^g$ for the leading part m' of the message. Now, factor:

$$c \equiv 2^{ge}(r2^{f-g} + m')^e \mod n \tag{4}$$

Therefore,

$$c/2^{ge} \equiv (r2^h + m')^e \mod n. \tag{5}$$

If r, h, m', and e are small enough that:

$$(r2^h + m')^e < n \tag{6}$$

Then $y = c2^{-ge} \mod n$ is a perfect e^{th} power of an integer, say $y = x^3$. Compute x as the the exact e^{th}:

$$x = \sqrt[3]{y} = \sqrt[3]{c/2^{ge} \mod n}. \tag{7}$$

Condition (6) can be met if the message m is long enough. Suppose n has bit-length 1024, which is $k = 128$ octets. The longest octet length allowed in PKCS #1 v. 1.5 for the message m is $k - 11 = 117$ octets, which is 936 bits.

Suppose that m has 670 trailing zero bits. The PKCS #1 v. 1.5 format arranges that the input the to raw private key exponentiation operation is bounded as follows:

$$r2^f + m < 3(2^{1008}) \tag{8}$$

Factoring out 2^{670} from trailing zero bits gives:

$$r2^h + m' = (r2^f + m)/2^{670} < 2^{1008-670}3 = 2^{338}3 \tag{9}$$

If $e = 3$, then condition (6) is met because

$$(r2^h + m')^3 < 2^{1014}27 < 2^{1019} < 2^{1023} < n. \tag{10}$$