Malicious Cryptography: Kleptographic Aspects

Adam Young[1] and Moti Yung[2]

[1] Cigital Labs
ayoung@cigital.com
[2] Dept. of Computer Science, Columbia University
moti@cs.columbia.edu

Abstract. In the last few years we have concentrated our research efforts on new threats to the computing infrastructure that are the result of combining malicious software (malware) technology with modern cryptography. At some point during our investigation we ended up asking ourselves the following question: what if the malware (i.e., Trojan horse) resides within a cryptographic system itself? This led us to realize that in certain scenarios of black box cryptography (namely, when the code is inaccessible to scrutiny as in the case of tamper proof cryptosystems or when no one cares enough to scrutinize the code) there are attacks that employ cryptography itself against cryptographic systems in such a way that the attack possesses unique properties (i.e., special advantages that attackers have such as granting the attacker exclusive access to crucial information where the exclusive access privelege holds even if the Trojan is reverse-engineered). We called the art of designing this set of attacks "kleptography." In this paper we demonstrate the power of kleptography by illustrating a carefully designed attack against RSA key generation.

Keywords: RSA, Rabin, public key cryptography, SETUP, kleptography, random oracle, security threats, attacks, malicious cryptography.

1 Introduction

Robust backdoor attacks against cryptosystems have received the attention of the cryptographic research community, but to this day have not influenced industry standards and as a result the industry is not as prepared for them as it could be. As more governments and corporations deploy public key cryptosystems their susceptibility to backdoor attacks grows due to the pervasiveness of the technology as well as the potential payoff for carrying out such an attack.

In this work we discuss what we call kleptographic attacks, which are attacks on black box cryptography. One may assume that this applies only to tamper proof devices. However, it is rarely that code (even when made available) is scrutinized. For example, Nguyen in Eurocrypt 2004 analyzed an open source digital signature scheme. He demonstrated a very significant implementation error, whereby obtaining a single signature one can recover the key [3].

In this paper we present a revised (more general) definition of an attack based on embedding the attacker's public key inside someone else's implementation of a

A.J. Menezes (Ed.): CT-RSA 2005, LNCS 3376, pp. 7–18, 2005.
© Springer-Verlag Berlin Heidelberg 2005

public-key cryptosystem. This will grant the attacker an exclusive advantage that enables the subversion of the user's cryptosystem. This type of attack employs cryptography against another cryptosystem's implementation and we call this kleptography. We demonstrate a kleptographic attcak on the RSA key generation algorithm and survey how to prove that the attack works.

What is interesting is that the attacker employs modern cryptographic tools in the attack, and the attack works due to modern tools developed in what some call the "provable security" sub-field of modern cryptographic research. From the perspective of research methodologies, what we try to encourage by our example is for cryptographers and other security professionals to devote some of their time to researching new attack scenarios and possibilities. We have devoted some of our time to investigate the feasibility of attacks that we call "malicious cryptography" (see [6]) and kleptographic attacks were discovered as part of our general effort in investigating the merger of strong cryptographic methods with malware technology.

2 SETUP Attacks

A number of backdoor attacks against RSA [5] key generation (and Rabin [4]) have been presented that exploit secretly embedded trapdoors [7–9]. Also, attacks have been presented that emphasize speed [1]. This latter attack is intended to work even when Lenstra's composite generation method is used [2] whereas the former three will not. However, all of these backdoor attacks fail when half of the bits of the composite are chosen pseudorandomly using a seed [7] (this drives the need for improved public key standards, and forms a major motivation for the present work). It should be noted that [1] does not constitute a SETUP attack since it assumes that a secret key remains hidden even after reverse-engineering.

We adapt the notion of a strong SETUP [8] to two games. For clarity this definition is tailored after RSA key generation (as opposed to being more general). The threat model involves three parties: the designer, the eavesdropper, and the inquirer.

The designer is a malicious attacker and builds the SETUP attack into some subset of all of the black-box key generation devices that are deployed. The goal of the designer is to learn the RSA private key of a user who generates a key pair using a device contained in this subset when the designer only has access to the RSA public keys. Before the games start, the eavesdropper and inquirer are given access to the SETUP algorithm in its entirety[1]. However, in the games they play they are not given access to the internals of the particular devices that are used (they cannot reverse-engineer them).

Assumptions: The eavesdropper and inquirer are assumed to be probabilistic poly-time algorithms. It is assumed that the RSA key generation algorithm is deployed in tamper-proof black-box devices. It is traditional to supply an RSA

[1] e.g., found in practice via the costly process of reverse-engineering one of the devices.

key generation algorithm with 1^k where k is the security parameter. This tells the generator what security parameter is to be used and assures that running times can be derived based on the size of the input. For simplicity we assume that the generator takes no input and that the security parameter is fixed. It is straightforward to relax this assumption.

Let D be a device that contains the SETUP attack.

Game 1: The inquirer is given oracle access to two devices A and B. So, the inquirer obtains RSA key pairs from the devices. With 50% probability A has a SETUP attack in it. A has a SETUP attack in it iff B does not. The inquirer wins if he determines whether or not A has the SETUP attack in it with probability significantly greater than $1/2$.

Property 1: (indistinguishability) The inquirer fails Game 2 with overwhelming probability.

Game 2: The eavesdropper may query D but is only given the public keys that result, not the corresponding private keys. He wins if he can learn one of the corresponding private keys.

Property 2: (confidentiality) The eavesdropper fails Game 1 with overwhelming probability.

Property 3: (completeness) Let (y, x) be a public/private key generated using D. With overwhelming probability the designer computes x on input y.

In a SETUP attack, the designer uses his or her own private key in conjunction with y to recover x. In practice the designer may learn y by obtaining it from a Certificate Authority.

Property 4: (uniformity) The SETUP attack is the same in every black-box cryptographic device.

When property 4 holds it need not be the case that each device have a unique identifier ID. This is important in a binary distribution in which all of the instances of the "device" will necessarily be identical. In hardware implementations it would simplify the manufacturing process.

Definition 1. *If a backdoor RSA key generation algorithm satisfies properties 1, 2, 3, and 4 then it is a* **strong SETUP**.

3 SETUP Attack Against RSA Key Generation

The notion of a SETUP attack was presented at Crypto '96 [7] and was later improved slightly [8]. To illustrate the notion of a SETUP attack, a particular attack on RSA key generation was presented. The SETUP attack on RSA keys from Crypto '96 generates the primes p and q from a skewed distribution. This

skewed distribution was later corrected while allowing e to remain fixed[2] [9]. A backdoor attack on RSA was also presented by Crépeau and Slakmon [1]. They showed that if the device is free to choose the RSA exponent e (which is often not the case in practice), the primes p and q of a given size can be generated uniformly at random in the attack. Crépeau and Slakmon also give an attack similar to PAP in which e is fixed. Crépeau and Slakmon [1] noted the skewed distribution in the original SETUP attack as well.

3.1 Notation and Building Blocks

Let $L(x/P)$ denote the Legendre symbol of x with respect to the prime P. Also, let $J(x/N)$ denote the Jacobi symbol of x with respect to the odd integer N.

The attack on RSA key generation makes use of the probabilistic bias removal method (PBRM). This algorithm is given below [8].

$PBRM(R, S, x)$:
input: R and S with $S > R > \frac{S}{2}$ and x contained in $\{0, 1, 2, ..., R-1\}$
output: e contained in $\{-1, 1\}$ and x' contained in $\{0, 1, 2, ..., S-1\}$
1. set $e = 1$ and set $x' = 0$
2. choose a bit b randomly
3. if $x < S - R$ and $b = 1$ then set $x' = x$
4. if $x < S - R$ and $b = 0$ then set $x' = S - 1 - x$
5. if $x \geq S - R$ and $b = 1$ then set $x' = x$
6. if $x \geq S - R$ and $b = 0$ then set $e = -1$
7. output e and x' and halt

Recall that a random oracle $R(\cdot)$ takes as input a bit string that is finite in length and returns an infinitely long bit string. Let $H(s, i, v)$ denote a function that invokes the oracle and returns the v bits of $R(s)$ that start at the i^{th} bit position, where $i \geq 0$. For example, if $R(110101) = 01001011110101...$ then,

$$H(110101, 0, 3) = 010$$

and

$$H(110101, 1, 4) = 1001$$

and so on.

The following is a subroutine that is assumed to be available.

$RandomBitString1()$:
input: none
output: random $W/2$-bit string
1. generate a random $W/2$-bit string str
2. output str and halt

Finally, the algorithm below is regarded as the "honest" key generation algorithm.

[2] For example, with $e = 2^{16} + 1$ as in many fielded cryptosystems.

*GenPrivatePrimes*1():
input: none
output: $W/2$-bit primes p and q such that $p \neq q$ and $|pq| = W$
1. for $j = 0$ to ∞ do:
2. $p = RandomBitString1()$ /* at this point p is a random string */
3. if $p \geq 2^{W/2-1} + 1$ and p is prime then break
4. for $j = 0$ to ∞ do:
5. $q = RandomBitString1()$
6. if $q \geq 2^{W/2-1} + 1$ and q is prime then break
7. if $|pq| < W$ or $p = q$ then goto step 1
8. if $p > q$ then interchange the values p and q
9. set $S = (p, q)$
10. output S, zeroize all values in memory, and halt

3.2 The SETUP Attack

When an honest algorithm *GenPrivatePrimes*1 is implemented in the device, the device may be regarded as an honest cryptosystem C. The advanced attack on composite key generation is specified by *GenPrivatePrimes*2 that is given below. This algorithm is the infected version of *GenPrivatePrimes*1 and when implemented in a device it effectively serves as the device C' in a SETUP attack.

The algorithm *GenPrivatePrimes*2 contains the attacker's public key N where $|N| = W/2$ bits, and $N = PQ$ with P and Q being distinct primes. The primes P and Q are kept private by the attacker. The attacker's public key is half the size of p times q, where p and q are the primes that are computed by the algorithm.

In hardware implementations each device contains a unique $W/2$-bit identifier ID. The IDs for the devices are chosen randomly, subject to the constraint that they all be unique. In binary distributions the value ID can be fixed. Thus, it will be the same in each copy of the key generation binary. In this case the security argument applies to all invocations of all copies of the binary as a whole.

The variable i is stored in non-volatile memory and is a counter for the number of compromised keys that the device created. It starts at $i = 0$. The variable j is not stored in non-volatile memory. The attack makes use of the four constants (e_0, e_1, e_2, e_3) that must be computed by the attacker and placed within the device. These quantities can be chosen randomly, for instance. They must adhere to the requirements listed in Table 1.

It may appear at first glance that the backdoor attack below is needlessly complicated. However, the reason for the added complexity becomes clear when the indistinguishability and confidentiality properties are proven. This algorithm effectively leaks a Rabin ciphertext in the upper order bits of pq and uses the Rabin plaintext to derive the prime p using a random oracle.

Note that due to the use of the probabilistic bias removal method, this algorithm is not going to have the same expected running time as the honest algorithm *GenPrivatePrimes*1(). The ultimate goal in the attack is to make it produce outputs that are indistinguishable from the outputs of an honest

Table 1. Constants used in key generation attack.

Constant	Properties
e_0	$e_0 \in \mathbb{Z}_N^*$ and $L(e_0/P) = +1$ and $L(e_0/Q) = +1$
e_1	$e_2 \in \mathbb{Z}_N^*$ and $L(e_2/P) = -1$ and $L(e_2/Q) = +1$
e_2	$e_1 \in \mathbb{Z}_N^*$ and $L(e_1/P) = -1$ and $L(e_1/Q) = -1$
e_3	$e_3 \in \mathbb{Z}_N^*$ and $L(e_3/P) = +1$ and $L(e_3/Q) = -1$

implementation. It is easiest to utilize the Las Vegas key generation algorithm in which the only possible type of output is (p, q) (i.e., "failure" is not an allowable output).

The value Θ is a constant that is used in the attack to place a limit on the number of keys that are attacked. It is a restriction that simplifies the algorithm that the attacker uses to recover the private keys of other users.

*GenPrivatePrimes*2():
input: none
output: $W/2$-bit primes p and q such that $p \neq q$ and $|pq| = W$
1. if $i > \Theta$ then output *GenPrivatePrimes*1() and halt
2. update i in non-volatile memory to be $i = i + 1$
3. let I be the $|\Theta|$-bit representation of i
4. for $j = 0$ to ∞ do:
5. choose x randomly from $\{0, 1, 2, ..., N - 1\}$
6. set $c_0 = x$
7. if $gcd(x, N) = 1$ then
8. choose bit b randomly and choose u randomly from $\mathbb{Z}_N^*$
9. if $J(x/N) = +1$ then set $c_0 = e_0^b e_2^{1-b} u^2 \ mod \ N$
10. if $J(x/N) = -1$ then set $c_0 = e_1^b e_3^{1-b} u^2 \ mod \ N$
11. compute $(e, c_1) = PBRM(N, 2^{W/2}, c_0)$
12. if $e = -1$ then continue
13. if $u > -u \ mod \ N$ then set $u = -u \ mod \ N$ /* for faster decr. */
14. let T_0 be the $W/2$-bit representation of u
15. for $k = 0$ to ∞ do:
16. compute $p = H(T_0||ID||I||j, \frac{kW}{2}, \frac{W}{2})$
17. if $p \geq 2^{W/2-1} + 1$ and p is prime then break
18. if $p < 2^{W/2-1} + 1$ or if p is not prime then continue
19. $c_2 = RandomBitString1()$
20. compute $n' = (c_1 \ || \ c_2)$
21. solve for the quotient q and the remainder r in $n' = pq + r$
22. if q is not a $W/2$-bit integer or if $q < 2^{W/2-1} + 1$ then continue
23. if q is not prime then continue
24. if $|pq| < W$ or if $p = q$ then continue
25. if $p > q$ then interchange the values p and q
26. set $S = (p, q)$ and break
27. output S, zeroize everything in memory except i, and halt

It is assumed that the user, or the device that contains this algorithm, will multiply p by q to obtain the public key $n = pq$. Making n publicly available is perilous since with overwhelming probability p can easily be recovered by the attacker. Note that c_1 will be displayed verbatim in the upper order bits of $n = n' - r = pq$ unless the subtraction of r from n' causes a borrow bit to be taken from the $W/2$ most significant bits of n'. The attacker can always add this bit back in to recover c_1.

Suppose that the attacker, who is either the malicious manufacturer or the hacker that installed the Trojan horse, obtains the public key $n = pq$. The attacker is in a position to recover p using the factors (P, Q) of the Rabin public key N. The factoring algorithm attempts to compute the two smallest ambivalent roots of a perfect square modulo N. Let t be a quadratic residue modulo N. Recall that a_0 and a_1 are ambivalent square roots of t modulo N if $a_0^2 \equiv a_1^2 \equiv t \ mod \ N$, $a_0 \neq a_1$, and $a_0 \neq -a_1 \ mod \ N$. The values a_0 and a_1 are the two smallest ambivalent roots if they are ambivalent, $a_0 < -a_0 \ mod \ N$, and $a_1 < -a_1 \ mod \ N$. The Rabin decryption algorithm can be used to compute the two smallest ambivalent roots of a perfect square t, that is, the two smallest ambivalent roots of a Rabin ciphertext.

For each possible combination of ID, i, j, and k the attacker computes the algorithm $FactorTheComposite$ given below. Since the key generation device can only be invoked a reasonable number of times, and since there is a reasonable number of compromised devices in existence, this recovery process is tractable.

$FactorTheComposite(n, P, Q, ID, i, j, k)$:
input: positive integers i, j, k with $1 \leq i \leq \Theta$
distinct primes P and Q
n which is the product of distinct primes p and q
Also, $|n|$ must be even and $|p| = |q| = |PQ| = |ID| = |n|/2$
output: $failure$ or a non-trivial factor of n
1. compute $N = PQ$
2. let I be the Θ-bit representation of i
3. $W = |n|$
4. set U_0 equal to the $W/2$ most significant bits of n
5. compute $U_1 = U_0 + 1$
6. if $U_0 \geq N$ then set $U_0 = 2^{W/2} - 1 - U_0$ /* undo the PBRM */
7. if $U_1 \geq N$ then set $U_1 = 2^{W/2} - 1 - U_1$ /* undo the PBRM */
8. for $z = 0$ to 1 do:
9. if U_z is contained in $\mathbb{Z}_N^*$ then
10. for $\ell = 0$ to 3 do: /* try to find a square root */
11. compute $W_\ell = U_z {e_\ell}^{-1} \ mod \ N$
12. if $L(W_\ell/P) = +1$ and $L(W_\ell/Q) = +1$ then
13. let a_0, a_1 be the two smallest ambivalent roots of W_ℓ
14. let A_0 be the $W/2$-bit representation of a_0
15. let A_1 be the $W/2$-bit representation of a_1
16. for $b = 0$ to 1 do:
17. compute $p_b = H(A_b||ID||I||j, \frac{kW}{2}, \frac{W}{2})$

18. if p_0 is a non-trivial divisor of n then
19. output p_0 and halt
20. if p_1 is a non-trivial divisor of n then
21. output p_1 and halt
22. output $failure$ and halt

The quantity $U_0 + 1$ is computed since a borrow bit may have been taken from the lowest order bit of c_1 when the public key $n = n' - r$ is computed.

4 Security of the Attack

In this section we argue the success of the attack and how it holds unique properties.

The attack is indistinguishable to all adversaries that are polynomially bounded in computational power[3]. Let C denote an honest device that implements the algorithm $GenPrivatePrimes1()$ and let C' denote a dishonest device that implements $GenPrivatePrimes2()$. A key observation is that the primes p and q that are output by the dishonest device are chosen from the same set and same probability distribution as the primes p and q that are output by the honest device. So, it can be shown that p and q in the dishonest device C' are chosen from the same set and from the same probability distribution as p and q in the honest device C[4].

In a nutshell confidentiality is proven by showing that if an efficient algorithm exists that violates the confidentiality property then either $W/2$-bit composites PQ can be factored or W-bit composites pq can be factored. This reduction is not a randomized reduction, yet it goes a long way to show the security of this attack.

The proof of confidentiality is by contradiction. Suppose for the sake of contradiction that a computationally bounded algorithm A exists that violates the confidentiality property. For a randomly chosen input, algorithm A will return a non-trivial factor of n with non-negligible probability. The adversary could thus use algorithm A to break the confidentiality of the system. Algorithm A factors n when it *feels* so inclined, but must do so a non-negligible portion of the time.

It is important to first set the stage for the proof. The adversary that we are dealing with is trying to break a public key pq where p and q were computed by the cryptotrojan. Hence, pq was created using a call to the random oracle R. It is conceivable that an algorithm A that breaks the confidentiality will make oracle calls as well to break pq. Perhaps A will even make some of the *same* oracle calls as the cryptotrojan. However, in the proof we cannot assume this. All we can assume is that A makes at most a polynomial[5] number of calls to the oracle and we are free to "trap" each one of these calls and take the arguments.

[3] Polynomial in $W/2$, the security parameter of the attacker's Rabin modulus N.

[4] The key to this being true is that n' is a random W-bit string and so it can have a leading zero. So, $|pq|$ can be less than W bits, the same as in the operation in the honest device before p and q are output.

[5] Polynomial in $W/2$.

Consider the following algorithm $SolveFactoring(N, n)$ that uses A as an oracle to solve the factoring problem.

$SolveFactoring(N, n)$:
input: N which is the product of distinct primes P and Q
n which is the product of distinct primes p and q
Also, $|n|$ must be even and $|p| = |q| = |N| = |n|/2$
output: $failure$, or a non-trivial factor of N or n
1. compute $W = 2|N|$
2. for $k = 0$ to 3 do:
3. do:
4. choose e_k randomly from $\mathbb{Z}_N^*$
5. while $J(e_k/N) \neq (-1)^k$
6. choose ID to be a random $W/2$-bit string
7. choose i randomly from $\{1, 2, ..., \Theta\}$
8. choose bit b_0 randomly
9. if $b_0 = 0$ then
10. compute $p = A(n, ID, i, N, e_0, e_1, e_2, e_3)$
11. if $p < 2$ or $p \geq n$ then output $failure$ and halt
12. if $n \ mod \ p = 0$ then output p and halt /* factor found */
13. output $failure$ and halt
14. output $CaptureOracleArgument(ID, i, N, e_0, e_1, e_2, e_3)$ and halt

$CaptureOracleArgument(ID, i, N, e_0, e_1, e_2, e_3)$:
1. compute $W = 2|N|$
2. let I be the Θ-bit representation of i
3. for $j = 0$ to ∞ do: /* try to find an input that A expects */
4. choose x randomly from $\{0, 1, 2, ..., N-1\}$
5. set $c_0 = x$
6. if $gcd(x, N) = 1$ then
7. choose bit b_1 randomly and choose u_1 randomly from $\mathbb{Z}_N^*$
8. if $J(x/N) = +1$ then set $c_0 = e_0^{b_1} e_2^{1-b_1} {u_1}^2 \ mod \ N$
9. if $J(x/N) = -1$ then set $c_0 = e_1^{b_1} e_3^{1-b_1} {u_1}^2 \ mod \ N$
10. compute $(e, c_1) = PBRM(N, 2^{W/2}, c_0)$
11. if $e = -1$ then continue
12. if $u_1 > -u_1 \ mod \ N$ then set $u_1 = -u_1 \ mod \ N$
13. let T_0 be the $W/2$-bit representation of u_1
14. for $k = 0$ to ∞ do:
15. compute $p = H(T_0||ID||I||j, \frac{kW}{2}, \frac{W}{2})$
16. if $p \geq 2^{W/2-1} + 1$ and p is prime then break
17. if $p < 2^{W/2-1} + 1$ or if p is not prime then continue
18. $c_2 = RandomBitString1()$
19. compute $n' = (c_1 \ || \ c_2)$
20. solve for the quotient q and the remainder r in $n' = pq + r$
21. if q is not a $W/2$-bit integer or if $q < 2^{W/2-1} + 1$ then continue
22. if q is not prime then continue

23. if $|pq| < W$ or if $p = q$ then continue
24. simulate $A(pq, ID, i, N, e_0, e_1, e_2, e_3)$, watch calls to R, and
    store the $W/2$-most significant bits of each call in list ω
25. remove all elements from ω that are not contained in $\mathbb{Z}_N^*$
26. let L be the number of elements in ω
27. if $L = 0$ then output $failure$ and halt
28. choose α randomly from $\{0, 1, 2, ..., L-1\}$
29. let β be the α^{th} element in ω
30. if $\beta \equiv \pm u_1 \ mod \ N$ then output $failure$ and halt
31. if $\beta^2 \ mod \ N \neq u_1^2 \ mod \ N$ then output $failure$ and halt
32. compute $P = gcd(u_1 + \beta, N)$
33. if $N \ mod \ P = 0$ then output P and halt
34. compute $P = gcd(u_1 - \beta, N)$
35. output P and halt

Note that with non-negligible probability A will not balk due to the choice of ID and i. Also, with non-negligible probability e_0, e_1, e_2, and e_3 will conform to the requirements in the cryptotrojan attack. So, when $b_0 = 0$ these four arguments to A will conform to what A expects with non-negligible probability. Now consider the call to A when $b_0 = 1$. Observe that the value pq is chosen from the same set and probability distribution as in the cryptotrojan attack. So, when $b_0 = 1$ the arguments to A will conform to what A expects with non-negligible probability. It may be assumed that A balks whenever e_0, e_1, e_2, and e_3 are not appropriately chosen without ruining the efficiency of $SolveFactoring$. So, for the remainder of the proof we will assume that these four values are as defined in the cryptotrojan attack.

Let u_2 be the square root of $u_1^2 \ mod \ n$ such that $u_2 \neq u_1$ and $u_2 < -u_2 \ mod \ n$. Also, let T_1 and T_2 be u_1 and u_2 padded with leading zeros as necessary such that $|T_1| = |T_2| = W/2$ bits, respectively. Denote by E the event that in a given invocation algorithm A calls the random oracle R at least once with either T_1 or T_2 as the $W/2$ most significant bits. Clearly only one of the two following possibilities hold:

1. Event E occurs with negligible probability.
2. Event E occurs with non-negligible probability.

Consider case (1). Algorithm A can detect that n was not generated by the cryptotrojan by appropriately supplying T_1 or T_2 to the random oracle. Once verified, A can balk and not output a factor of n. But in case (1) this can only occur at most a negligible fraction of the time since changing even a single bit in the value supplied to the oracle elicits an independently random response. By assumption, A returns a non-trivial factor of n a non-negligible fraction of the time. Since the difference between a non-negligible number and negligible number is a non-negligible number it follows that A factors n without relying on the random oracle. So, in case (1) the call to A in which $b_0 = 0$ will lead to a non-trivial factor of n with non-negligible probability.

Now consider case (2). Since E occurs with non-negligible probability it follows that A may in fact be computing non-trivial factors of composites n by

making oracle calls and constructing the factors in a straightforward fashion. However, whether or not this is the case is immaterial. Since A makes at most a polynomial number of calls[6] to R the value for L cannot be too large. Since with non-negligible probability A passes either T_1 or T_2 as the $W/2$ most significant bits to R and since L cannot be too large it follows that β and u_1 will be ambivalent roots with non-negligible probability. Algorithm A has no way of knowing which of the two smallest ambivalent roots $SolveFactoring$ chose in constructing the upper order bits of pq. Algorithm A, which may be quite uncooperative, can do no better than guess at which one it was, and it could in fact have been either. Hence, $SolveFactoring$ returns a non-trivial factor of N with non-negligible probability in this case.

It has been shown that in either case, the existence of A contradicts the factoring assumption. So, the original assumption that adversary A exists is wrong. This proves that the attack satisfies Property 2 of a SETUP attack.

Immediately following the test for $p = q$ in C and in C' it is possible to check that $gcd(e, (p-1)(q-1)) = 1$ and restart the entire algorithm if this does not hold. This handles the generation of RSA primes by taking into account the public RSA exponent e. This preserves the indistinguishability of the output of C' with respect to C.

5 Conclusion

Attacks on cryptosystems can occur from many different angles: a specification may be incorrect which requires provable security as a minimum requirement – preferably based on a complexity theoretic assumption and if not than on some idealization (e.g., assuming a random oracle like the idealization of unstructured one-way hash functions). However, implementations can have problems of their own. Here a deliberate attack by someone who constructs the cryptosystem (e.g., a vendor) has been demonstrated. This attack is not unique to the RSA cryptosystem and is but one of many possible attacks. However, it serves to demonstrate the overall approach. At a minimum, the message that we try to convey is that the scrutiny of code and implementations is crucial to the overall security of the cryptographic infrastructure, and if practitioners exercise scrutiny then we should be aware that we may need to completely trust each individual implementation to be correct in ways that may not be efficiently black-box testable (as our attack has demonstrated).

References

1. C. Crépeau, A. Slakmon. Simple Backdoors for RSA Key Generation. In *The Cryptographers' Track at the RSA Conference – CT-RSA '03*, pages 403–416, 2003.
2. A. K. Lenstra. Generating RSA Moduli with a Predetermined Portion. In *Advances in Cryptology – Asiacrypt '98*, pages 1–10, 1998.

[6] Polynomial in W.

3. P. Q. Nguyen. Can We Trust Cryptographic Software? Cryptographic Flaws in GNU Privacy Guard v1.2.3. In *Advances in Cryptology – Eurocrypt '04*, pages 555–570, 2004.
4. M. Rabin. Digitalized signatures and public-key functions as intractable as factorization. TR-212, MIT Laboratory for Computer Science, January 1979.
5. R. Rivest, A. Shamir, L. Adleman. A method for obtaining Digital Signatures and Public-Key Cryptosystems. In *Communications of the ACM*, volume 21, n. 2, pages 120–126, 1978.
6. A. Young, M. Yung. "Malicious Cryptography: Exposing Cryptovirology," Wiley Publishing Inc., Feb. 2004.
7. A. Young, M. Yung. The Dark Side of Black-Box Cryptography, or: Should we trust Capstone? In *Advances in Cryptology – Crypto '96*, pages 89–103, 1996.
8. A. Young, M. Yung. Kleptography: Using Cryptography Against Cryptography. In *Advances in Cryptology – Eurocrypt '97*, pages 62–74, 1997.
9. A. Young. Kleptography: Using Cryptography Against Cryptography. PhD Thesis, Columbia University, 2002.