OpenSSL 기반 RSA서버에 대한 Timing Attack 구현

홍정대° 박근수
서울대학교 전기,컴퓨터공학부
{jdhong°, kpark}@theory.snu.ac.kr

An implementation of the timing attack on OpenSSL-based RSAserver

Jeongdae Hong° Kunsoo Park
School of Computer Science & Engineering, Seoul National University

1996년 P. Kocher에 의해 시차공격(Timing attack)이 제안된 후 일반적인 RSA 구현 시 시간차를 줄이기 위해 중국인의 나머지 정리와 Montgomery 알고리즘과 같은 다양한 방법들이 적용되어왔다. 2003년 D. Brumley 와 D. Boneh가 OpenSSL[2]에서 구현된 RSA 알고리즘을 분석하여 시차공격[3]이 가능함을 보였다. 본 논문은 이들의 방법을 OpenSSL을 기반으로 하는 서버를 대상으로 구현한 실험 결과를 보인다.

1 서 론

시차공격은 부채널 공격(side-channel cryptanalysis)의 일종으로 암호 알고리즘의 수학적 논의를 배제하고 연산과정에서 필연적으로 발생하는 시간차이를 이용하여 암호가 구현된 장비나 시스템을 공격하는 방법이다.

Kocher는 RSA 암호화 및 복호화 과정에서 공통으로 적용되는 멱승연산에서 지수의 비트가 1일 때와 0일 때 연산시간이 다르다는 사실을 이용하여 비밀키를 찾아내는 방법을 제안[1,4]하였고, Schindler는 중국인의 나머지 정리(Chinese Remainder Theorem)와 Montgomery 알고리즘[6] 등이 적용된 구현에서 시차공격 방법[5]으로 인수분해가 가능함을 보였다.

2003년 D. Brumley 와 D. Boneh는 RSA구현에 중국인의 나머지 정리, Montgomery 알고리즘, Sliding windows 멱승연산, Karatsuba 곱셈 등이 적용된 OpenSSL을 분석해서 시차공격이 가능함을 보였다.

본 논문은 OpenSSL을 기반으로 하는 RSA서버를 공격자 컴퓨터 내부(interprocess)와 외부 컴퓨터(LAN)에 설치하고 이들의 방법을 사용하여 시차공격한 실험결과를 보인다.

2 OpenSSL에 구현된 RSA 분석

2.1 OpenSSL의 RSA 구현

RSA[7]는 암호화 및 복호화 과정에 동일한 모듈러 멱승연산을 실시한다.

$$m = c^d \bmod N$$

($N=qp$: RSA modulus, d : 비밀키, c : 암호문)

OpenSSL은 멱승연산에 중국인의 나머지 정리(이하 CRT)를 사용한다. 즉 $m = c^d \bmod N$를 계산하는데 먼저 $m_1 = c^{d1} \bmod p$ 와 $m_2 = c^{d2} \bmod q$ 로 나누어 계산한 후 m_1과 m_2 를 CRT방법으로 결합하여 m 을 계산한다. $c^{d1} \bmod p$ 와 $c^{d2} \bmod q$ 계산에는 동일한 연산이 적용되므로 모듈러 멱승연산은 이후 $g^d \bmod q$로 설명한다.

$g^d \bmod q$ 계산은 Sliding windows 멱승연산[8] 방법을 사용하는데 단순한 square & multiply 알고리즘 ($\log_2 d$ 번의 제곱연산, $\log_2 d/2$ 번의 곱셈 연산 필요)과 달리 d 를 windows크기를 기준으로 나눠 windows에 포함된 비트 1의 개수 만큼 제곱연산을 수행한 다음 미리 계산해둔 곱셈표에서 windows에 해당하는 값을 찾아 곱하는 방법이다. 이 방법을 사용하면 g가 곱해지는 횟수가 단순한 square & multiply 알고리즘에 비해

$$E[\ \#\ \text{multiply by } g] \approx \log_2 d / 2^{w-1}(w+1)$$

(d : 비밀키, w : windows size)

로 줄어든다. OpenSSL에서는 1024-비트 모듈러에 대한 windows 크기로 5를 사용한다.

Sliding windows 멱승연산도 매 단계에서 모듈러 곱셈을 수행하는데 이때 Montgomery 알고리즘을 적용한다. Montgomery 알고리즘은 컴퓨터에서의 모듈러 곱셈 연산에서 모듈러 q 대신 $R = 2^k$ ($2^{k-1} \le q < 2^k$)을 사용하여 연산 속도를 높이는 방법이다. $x * y \bmod q$를 계산하기 전 x와 y를 각각 $xR \bmod q$와 $yR \bmod q$ 형태로 변경하여 xyR^2를 계산한 후 R^{-1}를 곱해 $xyR^2 * R^{-1} = xyR \bmod q$를 구한다. 그리고 계산값이 모듈러 q보다 크면 추가로 q를 빼서 그 값이 $[0,q)$ 구간에 있도록 한다. Schindler는 $g^d \bmod q$ 에서 g가 q의 배수에 가까워짐에 따라 Montgomery 알고리즘의 추가적인 뺄셈의 횟수가 늘어나다가 q의 배수를 넘는 순간 급격히 줄어든다는 사실을 발견[5]하였다.

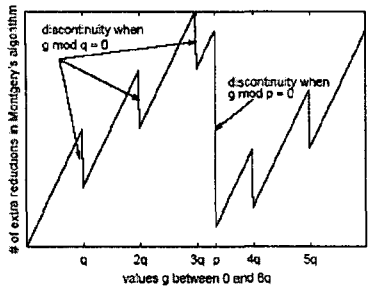

그림 1. g값에 따른 추가적인 뺄셈 개수 변화

OpenSSL은 RSA 연산에서 두 가지 곱셈연산을 사용하는데 곱해지는 두 수의 길이가 같으면 Karatsuba 알고리즘[9]을 사용하고 다르면 일반적인 곱셈을 사용한다. 비슷한 길이의 숫자를 곱하는데 일반적인 곱셈을 사용하면 $O(n^2)$의 시간이 소요되는데 비해, Karatsuba 방법을 사용하면 $O(n^{1.58})$의 시간이 걸려 보다 빠른 연산속도를 보인다.

2.2 RSA 연산의 시간차 분석

앞에서 살펴본 바와 같이 OpenSSL은 Montgomery 알고리즘과 Karatsuba알고리즘에서 모듈러 q를 전후한 값에서 시간차가 발생한다. 하지만 두 알고리즘에 의한 시간차는 서로 상쇄된다.

g가 q에 가까워질수록 Montgomery 알고리즘의 추가적인 뺄셈의 횟수가 늘어났다가 q를 넘는 순간 급격히 줄어들기 때문에 $g < q$면 $g > q$때 보다 연산 시간이 더 걸린다.

반면 g가 q에 거의 근접하면 곱셈연산으로 Karatsuba 알고리즘이 적용되다가 g가 q를 넘은 직후에는 일반적인 곱셈연산이 수행된다. 즉 곱셈연산에 의하면 $g < q$때 $g > q$ 보다 연산 시간이 덜 걸린다.

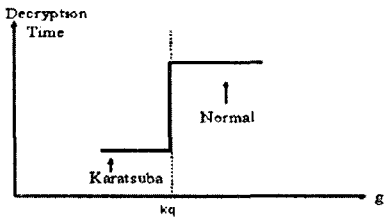

그림 2. 곱셈 연산에 따른 복호화 시간

3 Brumley & Boneh's 방법

Brumley & Boneh는 RSA 모듈러 N의 인수(factor)를 알면 비밀키 d를 알 수 있다는 사실[10]을 바탕으로 g를 인수 q에 근사(approximation)시켜 가는 방법을 사용한다. 인수 q의 최상위 비트부터 하위 비트로 근사 시키는데 인수 q의 절반까지 결정한 후에는 Coppersmith 알고리즘[11]을 적용하여 인수 분해한다.

공격자는 먼저 인수 q의 최상위 2-3 비트를 추측하는데 가능한 모든 비트 조합을 g로 놓고 복호화 시간을 살펴본다. 시간이 많이 걸리는 두 수가 각각 p와 q인데 이중 첫 번째 것을 q로 놓고 이를 확장시켜 나간다.

(1) 인수 q의 i-1 번째 비트까지 찾았다면 새롭게 추측하는 g는 i-1 번째 비트까지는 인수 q와 같게 놓고 나머지 비트를 0이라고 가정한 후 i 번째 bit를 찾는다.

(2) g_{hi} 를 i-1 번째 비트까지는 g와 같으면서 i 번째 비트가 1인 숫자로 놓는다. 실제 인수 q에서 i 번째 비트가 1 이면 $g < g_{hi} < q$이고, 0 이면 $g < q < g_{hi}$ 이다.

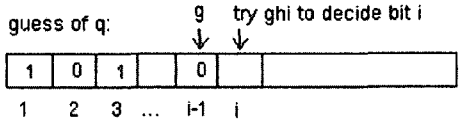

그림 3. i번째 비트 결정 과정

(3) 모듈러 멱승연산 과정에 Montgomery 형태의 숫자를 일반적인 형태로 변경하는 연산($u_gR = g$, $u_{ghi}R = g_{hi}$)이 수행되므로 $u_g = gR^{-1} \bmod N$ 과 $u_{ghi} = g_{hi}R^{-1} \bmod N$ 을 계산한다.

(4) 서버에게 u_g 와 u_{ghi} 의 복호화를 요청해서 그 소요시간 t_1 = DecryptTime(u_g) 과 t_2 = DecryptTime(u_{ghi})을 측정한다.

(5) t_1 과 t_2 의 시간차 $\triangle = |t_1 - t_2|$ 를 계산하여 그 차가 크면 $g < q < g_{hi}$ 관계가 성립하므로 q의 i 번째 비트를 0 으로, 작으면 $g < g_{hi} < q$이므로 1 로 결정한다.

4 실험 결과

4.1 실험 환경

실험에 사용된 운영체제는 windows 2000이고 프로세서는 Intel Pentium 4 2.4GHz, 주 기억장치는 1Gbyte인 하드웨어가 사용되었으며 OpenSSL은 시차공격에 대한 보안 패치가 되기 전 버전인 0.9.7을 사용하였고, RSA키는 1024비트로 OpenSSL의 RSA_generate_key()함수로 생성하였다. 네트워크 연결은 1Gb 3Com 스위치를 사용하였다. 또한 정밀한 시간 측정이 요구되므로 시간 측정에는 인텔에서 제공하는 RDTSC[12]를 사용하였다

4.2 실험 방법

Sliding windows 에서 추측값 g의 곱셈 횟수가 줄어든 것을 보상하기 위해 추측값에 1씩 더해가면서 $g+1, g+2, \ldots, g+n$ 의 이웃하는 숫자를 만들고 이를 복호화시킨 시간을 더하여 $T_g = \sum$ DecryptTime($g+1, g+2, \ldots, g+n$) 를 구한다. 또한 각각의 추측값에 대한 복호화 시간도 측정 횟수를 증가시킬수록 정확한 값이 되므로 하나의 추측값에 대한 복호화 시간은 최소 9회 측정한 값의 평균으로 하였다.

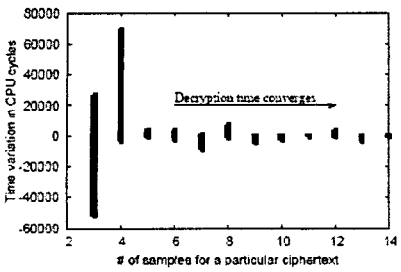

그림 4. 복호화 횟수에 따른 시간 변화

(1) 단일 프로그램내에 OpenSSL기반 RSA서버와 공격자를 두고 인수 q를 찾았다.

(2) 동일 컴퓨터에 서버 프로그램과 공격자 프로그램을 별도 동작(interprocess)시켜 인수 q를 찾았다.

(3) 네트워크(LAN)로 연결된 두 대의 컴퓨터에 TCP서버와 공격자를 별도 동작시켜 인수 q를 찾았다.

4.3 실험 결과

그림 5는 실험(1) 이웃 400일 때 인수 q의 3에서 256번째 비트까지 추측값에 따른 시간차를 나타내고 있으며

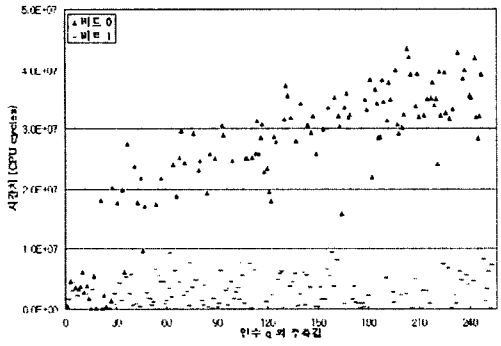

그림 5. 인수 q의 추측값에 따른 비트별 시간차

표 1은 인수 q의 129에서 161번째 비트에서 시간차에 대한 평균을 나타내고 있다. 각 환경에서 Brumley & Boneh의 방법이 성공적으로 적용되며 이웃하는 숫자의 개수가 많으면 비트 0과 1일 때의 시간차도 커진다는 사실을 확인하였다.

시간차 Δ(cycles)		실험 1	실험 2	실험 3
이웃 400	비트 0	3.6E+7	4.5E+7	7.0E+6
	비트 1	3.6E+6	1.1E+7	1.1E+6
이웃 800	비트 0	2.6E+7	5.2E+7	1.6E+7
	비트 1	9.5E+5	1.6E+6	3.6E+6

표 1. 이웃 400 / 800에서 비트에 따른 시간차

5 결론

OpenSSL을 기반으로 동작되는 RSA서버에 대하여 단일 프로그램, 동일 컴퓨터, 네트워크 환경에서 Brumley & Boneh의 방법을 사용하여 시차공격이 가능함을 보았다. 하지만 이들의 방법은 OpenSSL에서 2003년 3월 이후 제공하는 시차공격에 대한 패치(blinding)를 설치하거나 Montgomery 알고리즘에서 추가적인 뺄셈이 없도록 설계된 환경에서는 사용할 수 없다. 그러나 시차공격을 고려하지 않고 RSA 구현에 OpenSSL과 유사한 알고리즘들을 사용한 시스템이나 소프트웨어에 대해서는 Brumley & Boneh의 방법이나 이와 유사한 시간차 분석에 의한 공격이 가능할 것으로 보인다.

6 참고 문헌

[1] Paul Kocher. Timing attacks on implementations of differ-hellman, RSA, DSS, and other systems. *Advances in Cryptology*, pages 104-113, 1996.

[2] OpenSSL Project. Openssl. http://www.openssl.org

[3] David Brumley, Dan Boneh. Remote timing attacks are practical. *12th Usenix Security Symposium*, 2003

[4] Jean-Francois et al. A practical implementation of the timing attack. In *CARDIS*, pages 167-182, 1998..

[5] Wener Schindler. A timing attack against RSA with the Chinese remainder theorem. In *CHES* 2000, pages 109-124, 2000.

[6] Peter Montgomery, Modular multiplication without trial division. *Mathematics of computation*, 44(170): 519-521,1985.

[7] R. Rivest, A. Shamir, L. Adleman. A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM, pages 120-126, 1978.

[8] L. C. K. Hui and K.-Y. Lam. Fast square-and-multiply exponentiation for RSA. *Electronics Letters*, 30(17): 1396-1397, August 1994.

[9] A. Karatsuba and Y. Ofman. Multiplication of Multidigit Numbers on Automata. *Soveit physics- Doklady*, 7, 595-596, 1963.

[10] Dan Boneh. Twenty Years of Attacks on the RSA Cryptosystem. In Notices of the American Mathematical Society (AMS), Vol. 46, No. 2, pp. 203--213, 1999

[11] D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10:233-260, 1997.

[12] Intel. Using the RDTSC instruction for performance monitoring. Technical report, 1997.