Attacking OpenSSL using Side-channel Attacks: the RSA case study

Praveen Kumar Vadnala, Łukasz Chmielewski

Riscure BV, Delft, The Netherlands

Abstract. We show that RSA implementation present in OpenSSL can be successfully attacked using side-channels. In OpenSSL, the modular exponentiation is implemented using *m-ary* method, where a table of size $2m$ entries is precomputed. The exponent is divided into words of m-bits each and the algorithm proceeds one word at a time using the precomputed table. Furthermore, to protect against side-channel attacks, it implements message blinding countermeasure.
However, this implementation has the following vulnerability: if two operations share the same secret key bit, the correlation between these samples will be higher compared to independent bits. This fact can be used to differentiate the secret key bit from 0 and 1 and can be exploited to reveal the secret key in a so called *cross-correlation attack*. The cross-correlation attack works even in the case of message blinding because it does not depend on the absolute values of the operands. In the case of OpenSSL implementation, we cross-correlate the precomputations with the operands used in m-ary exponentiation to recover the full secret key.
Furthermore, we explore other, more advanced ways to attack OpenSSL RSA, so called single traces attacks: template and horizontal attacks in particular.

Keywords: Side-channel attacks, RSA, OpenSSL, cross-correlation attack.

Side-channel attacks. Implementations of cryptographic algorithms leak information about the secret key which could potentially be exploited by an attacker. Examples of such leakages include timing [Koc96], power consumption [KJJ99], and electromagnetic emission [AARR03]. These so called *side-channel attacks* are very powerful in the sense that one can completely break the security of cryptographic devices with a very inexpensive setup. In a Simple Power Analysis (SPA) attack the attacker tries to recover the secret key by visually inspecting the power measurements from the cryptographic device. For example, if we consider the square-and-multiply method of implementing the exponentiation, the pattern in the power trace corresponding to the square operation (exponent bit 0) will be different from the pattern corresponding to square and multiply operations (exponent bit 1). By observing one or few such traces from the measurement, the attacker could recover the full secret key. On the contrary, Differential Power Analysis (DPA) attacks require several traces but little knowledge about the cryptographic implementations and hence can be very powerful. It is well known that straightforward implementations of RSA can be successfully attacked using Simple Power Analysis (SPA) as well as DPA attacks [KJJ99].

Countermeasures. To counteract DPA attacks, several countermeasures have been proposed in the literature. Masking, besides hiding is one of the most widely used countermeasure to prevent side-channel attacks [CJRR99]. The basic idea behind masking is that each sensitive variable (a function of known variable and the secret key) used in the algorithm is split into two shares, one of which is generated randomly (called *mask*), while the second share is computed using the mask and the sensitive variable. All the subsequent operations in the algorithm are applied separately on the shares, which are combined at the end to produce the desired ciphertext (or plaintext).

OpenSSL RSA. RSA requires modular exponentiation modulo very large numbers, which is a very costly operation. To improve the efficiency, OpenSSL employs several optimizations. In a basic implementation, computing $m^d \mod n$ is achieved by using *binary exponentiation* algorithm. Here, each bit in d is processed from left-to-right (or right-to-left) and based on the bit value we either perform only square (if bit is 0) or square-and-multiply (if the bit is 1). This idea can actually be extended to a window of size k where $w = 2^k$. In this so called m-ary method, we precompute the values $m^0, m^1, \cdots, m^{w-1}$ and store in a table. Next, we represent the exponent d in *base-w* system. We then scan the *base-w* represented exponent from left-to-right one word at a time and multiply the intermediate result with the appropriate precomputed result from the table.

To further improve the efficiency, OpenSSL also implements RSA using Chinese Reminder Theorem (referred to as RSA-CRT). Let the primes be p, q and $n = pq$. We precompute the following:

$$d_p = d \mod p - 1$$
$$d_q = d \mod q - 1$$
$$q_{inv} = q^{-1} \mod p$$

We can now compute $m = c^d \mod n$ as:

$$\begin{aligned} m_1 &= c^{d_p} \mod p \\ m_2 &= c^{d_q} \mod q \\ h &= q_{inv}(m_1 - m_2) \mod p \\ m &= m_2 + hq \end{aligned}$$

Though we need to compute two modular exponentiations in this case, this method is more efficient as we operate on smaller exponents and moduli.

To counteract against SCA, OpenSSL also implements two countermeasures: message blinding and multiply-always-exponentiation. In message blinding, we generate a new secret random r for each encryption. The message m is then represented using two shares: $m_1 = r^e \mod n$ and $m_2 = r^{-1} \mod n$, where e is the public key. On the other hand, multiply-always countermeasure tries to counter SPA by keeping the sequence of operations constant irrespective of the value of the bit. Namely, the square-and-multiply operation is performed for all the bits but the result after multiplication is discarded when the value of the bit is 0.

Cross-correlation attack. When the exponentiation is implemented using multiply-always algorithm it is not possible to differentiate the secret key bits from 0 and 1. We recall the multiply-always binary exponentiation algorithm in Algorithm 1. Here we see four types of consecutive operations: square|multiply (SM), square|multiply|discard (SMd), multiply|square (MS), and multiply|discard|square(MdS). From the algorithm, we can see that SM, SMd, and MS do not share the operands but MdS does. We can use the fact that MdS and MS differ by their shared operand to perform a correlation attack. By calculating the correlation between the points corresponding to multiplication and square operation, we can differentiate between MdS and MS operations. Namely, in case of MdS, the correlation is stronger compared to MS as the former shares an operand between square and multiplication. This can be used to distinguish between bits 0 and 1. The similar attack can also be applied to an implementation which uses m-ary method by correlating the leakages from precomputations with the actual exponentiations. Furthermore, this attack works even in case of message blinding as it does not depend on the absolute values of the operands.

Algorithm 1 Multiply always binary exponentiation algorithm

Require: Message m, exponent d and modulus n
Ensure: $m^d \mod n$

```
1: s ← 1
2: for i := |d| − 1 to 0 do
3:     s ← s.s mod n                          ▷ Square
4:     if d_i == 1 then
5:         s ← s.m mod n                      ▷ Multiply
6:     else
7:         t ← s.m mod n                      ▷ Multiply and discard
8:     end if
9: end for
10: return s
```

Single Trace Attacks. Horizontal attacks on RSA [Wal01] are emerging forms of side-channel attacks on exponentiation-based or scalar-multiplication-based algorithms. Their methodology allows recovering the exponent bits through the analysis of individual traces.

Template attacks [CRR03] are similar to the horizontal attacks but they require profiling. In this case, a cryptographic device having a fixed or known key is used to learn the statistics (for instance, mean and variance) related to the processing of known exponent bits. Subsequently, a single trace is attacked using the gathered statistic.

Our results. We ported the OpenSSL RSA implementation to our custom-made board consisting of high-end Cortex-M4 CPU which runs at 168 MHz. We measured the power consumption of the board while it performs several RSA decryptions. We then applied several signal processing techniques to improve the signal-to-noise ratio as well as align the traces. Next, we identified the windows corresponding to the precomputations and the actual modular exponentiation. By correlating the samples from the preprocessing with the samples corresponding to the exponentiation, we could recover the two exponents completely.

Furthermore, we analyze the resistance of the OpenSSL to more advanced sophisticated attacks, so called single traces attacks: template and horizontal attacks, in particular.

References

[AARR03] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM Side-Channel(s). In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop Proceedings*, volume 2523 of *Lecture Notes in Computer Science*, pages 29–45. Springer, 2003.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *CRYPTO*, 1999.

[CRR03] Suressh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *LNCS*, pages 13–28. Springer, 2003.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO 1999, 19th Annual International Cryptology Conference Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO 1996 Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[Wal01] Colin D. Walter. Sliding windows succumbs to Big Mac attack. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *LNCS*, pages 286–299. Springer, 2001.