Side-Channel Attack against RSA Key Generation Algorithms

Aurélie Bauer, Eliane Jaulmes, Victor Lomné,
Emmanuel Prouff, and Thomas Roche

ANSSI,
51, Bd de la Tour-Maubourg, 75700 Paris 07 SP, France
{firstname.lastname}@ssi.gouv.fr

Abstract. Many applications of embedded devices require the generation of cryptographic secret parameters during the life cycle of the product. In such an unsafe context, several papers have shown that key generation algorithms are vulnerable to side-channel attacks. This is in particular the case of the generation of the secret prime factors in RSA. Until now, the threat has been demonstrated against naive implementations whose operations' flow depends on secret data, and a simple countermeasure is to avoid such kind of dependency. In this paper, we propose a new attack that renders this defence strategy ineffective. It is in particular able to break secure implementations recommended by the ANSI X9.31 and FIPS 186-4 standards. We analyse its efficiency for various realistic attack contexts and we demonstrate its practicality through experiments against a smart-card implementation. Possible countermeasures are eventually proposed, drawing the following main conclusion: prime generation algorithms should avoid the use of a prime sieve combined with a deterministic process to generate the prime candidates from a random seed.

1 Introduction

When signing or decrypting with RSA it is nowadays well-known that the modular exponentiation must be implemented with care to defeat Side-Channel Attacks (SCA). The use of the secret exponent indeed induces some vulnerabilities and a wide number of studies have been dedicated to this specific operation [4,7,12,20,23,27]. However this is not the unique vulnerable step of RSA cryptosystem implementations. The prime generation algorithm aiming at finding two large prime factors p and q to build the RSA modulus can also be threatened by SCA. Until recent years, this computation was solely performed during the device personalisation (when the device is uniquely associated to a device holder) and, for this reason, SCA was considered to be out-of-scope. This is no longer the case: the arrival of new security services (mobile payment, e-ticketing, OTP generations, and so on) has raised the need for devices able to perform key generations during their life cycle. The RSA key generation has then left the safe context of production firms for an hostile environment. This assessment has

L. Batina and M. Robshaw (Eds.): CHES 2014, LNCS 8731, pp. 223–241, 2014.

been highlighted in several papers [8,16,28] which show that the key generation security must be taken into account for today *open* platforms.

Prime Generation Algorithms. A straightforward method to generate a large prime number is to start from a random value, to perform a *provable primality* test, and in case of an invalid answer, to repeat the process with another random data until a prime is found. This procedure obviously leads to a valid solution, but also provides very costly prime generations. In fact, *provable prime* generations are considered to be less efficient, in time and memory usage, than *probable prime* ones [6]. Indeed, using the latter consists in replacing the costly primality proof of the selected candidate by a series of relatively efficient probabilistic tests [21]. When correctly parametrised, this probabilistic approach provides a satisfying confidence level in the primality of the generated value. Still this technique may remain costly, especially for embedded systems, since almost all probabilistic primality tests are based on non trivial arithmetic operations over large integer rings. For this reason, probable prime generation algorithms are often implemented together with a *prime sieve* [6,21]. That way, each new prime candidate is first checked for small factors (up to a fixed bound) by successive divisibility tests, and can thus be possibly eliminated without having to go through the probabilistic primality tests.

Whereas the implementations discussed previously enable to check whether candidates are prime or not at moderate cost, the overall efficiency of the algorithm can still remain poor if no particular attention is paid to the "generation" phase. In particular, randomly generating each new candidate until a probable prime is found turns out to be hardly practical. This is especially true when the access to the random number generator is expensive, which is usually the case on embedded devices. A more efficient technique consists in calling the random generator only once and in using the obtained value as a seed to generate a succession of prime candidates in a deterministic way. Usually the seed is simply chosen to be odd and incremented by an even constant iteratively [6,5], but any other kind of deterministic process can be devised. Early studies on the probable prime generations implemented with a prime sieve and an incremental generation of candidates [6,5] exhibit efficient optimisations and show that the entropy of the generated primes is close to the maximum. Therefore, even though recent work have proposed interesting alternatives [11] or discussed the relevance of the entropy evaluation [18], the approach with a prime sieve and deterministic candidates generation turns out to be nowadays the most common procedure in constrained environments. It is actually recommended by international standards like ANSI X9.31 [1] and FIPS 186-4 [17].

Attacking the Sieving Process. In [16], Finke *et al.* observe that using a deterministic process to generate the sequence of candidates from a random seed, combined with a naive implementation of the prime sieve, is threatened by a Simple Power Analysis (SPA for short). Roughly, if a side-channel attacker is able to identify each divisibility test on a leakage trace and if the sieving process abort as soon as a small factor is found, then a simple equation system can be obtained, whose resolution brings information on the generated prime (see [16] for more details). The type of weakness identified in [16] can potentially be found in any algorithm processing a prime sieve whose flow of operations is data dependent (which is for instance the case of naive implementations of the X9.31 standard [1]). To avoid it, a simple (and usually fairly efficient) countermeasure hence amounts to balance the conditional branches in the implementation. One way to do so is to apply the prime sieve entirely even if, at some point, the algorithm highlights a divisibility (in other words, the prime sieve should not be stopped once a divider is found). It must be observed that this implementation choice does not only prevent the state-of-the-art attacks but, as discussed in [6], also leads to a significant efficiency gain.

Results. This paper focuses on the security of the probable prime generation algorithms discussed previously (with prime sieve and deterministic candidates generation). For such algorithms, which, to the best of our knowledge, correspond to the most efficient and up-to-date implementations met on embedded devices, we exhibit an *Advanced Side-Channel Analysis* on the sieving process even when the latter is implemented to defeat the state-of-the-art attacks [16]. Contrary to [28], our attack does not target the probable prime tests but the prime sieve which was believed to be safe if implemented in a regular way. We show how useful information can be extracted from the divisibility phase and how this could finally lead, for practical implementations, to the recovery of more than half bits of information on the prime number generated. Combined with a well-known lattice reduction technique due to Coppersmith [14,3], we show that the attack leads to the recovery of a 1024-bit RSA modulus. Moreover it severely undermines the security of larger moduli. Additionally to the theoretical analysis, we provide experimental results from the analysis of the side-channel leakage on a real device. The success of these experimentations highlights the practicality of our attack and, as a side effect, shows that countermeasures against SPA attacks are not sufficient to ensure security. Our work also shows that the use of a deterministic process to build a sequence of candidates from a random seed represents a serious weakness. In view of this, the non-deterministic candidates generation proposed by Fouque and Tibouchi [18] seems to be a good alternative. As argued by the authors, it would moreover increase the entropy of the generated probable primes. Another possibility could be to implement the provable prime generation algorithm proposed recently in [11].

2 On a Standard Prime Generation Implementation

This section aims at describing the design of a standard RSA prime generation algorithm, such as recommended by the norms ANSI X9.31 [1, Annexes B and E] and FIPS 186-4 [17, Annex C]. This description is completed with implementation details that must be considered when embedding such algorithms on constrained devices. Implementation choices are also made and strengthened by efficiency rationales. Eventually, the section ends with an implementation of a probable prime generation algorithm which is very close to what can be found in todays' industry of embedded devices.

2.1 A Prime Generation Algorithm for Constrained Environments

The purpose of probable prime generation algorithms is to return a number which satisfies a series of probabilistic tests and is indistinguishable from a random prime. The latter property is ensured by randomly generating the candidates on which the probabilistic tests are passed. For efficiency reasons however, the implementations discussed here (and recommended in ANSI X9.31 [1, Annexes B and E] and FIPS 186-4 [17, Annex C]) do not generate all the candidates at random but deduce them from a common random seed through a deterministic process. In the following, we assume that the latter simply consists in adding a multiple of a constant, but our analysis would hold for any other deterministic process. Eventually, to spare the use of the costly probabilistic tests, a prime sieve is applied to directly eliminate candidates with small prime factors. More details about these two steps are given hereafter.

Probabilistic primality tests. Testing the primality of a candidate is usually done using Miller-Rabin and Lucas probabilistic tests. The reader can refer to [22] for their description. Actually, the only important fact to mention is that Miller-Rabin test performs several dozens of exponentiations of the form $a^{t \cdot 2^s} \mod v$, for a a random number and v the tested candidate[1]. As v is large, such exponentiations are very costly and are usually performed thanks to a modular arithmetic co-processor.

Prime sieve. The purpose of the prime sieve is to reduce the number of Miller-Rabin's tests. It precedes them and eliminates the candidates having small factors. It consists in a divisibility test w.r.t. all primes lower than some bound r. For efficiency reasons, a classical choice is to select only primes lower than 256 (there are 53 such primes). This choice indeed has both the advantage to limit the size of the array containing the sieve elements and to get efficient divisions even for an 8-bit architecture with limited instructions set. By Mertens' Theorem[2] [25], one can prove that choosing r as 256 enables to eliminate around 87.5% of the tested integers without executing the probabilistic tests. On the other hand increasing r to 9-bit long primes, "only" allows to exclude an additional 1.4% of the integers. Together with the efficiency reasons, this poor discrimination gain explains why the choice $r = 256$ is sound for prime sieves in constrained environments.

Summing-up all these steps leads to a full implementation of a standard prime generation algorithm on constrained environment, see Algorithm 1.

[1] In these relations, the parameters s and t satisfy $v - 1 = 2^s \cdot t$ and t is odd.

[2] The probability that a random integer is not divisible by a number smaller than r is well approximated by $1/\log(r)$.

Algorithm 1. Prime Generation Algorithm (for constrained environments)

```
   Input   : A bit-length $\ell$, an even constant $\tau$, the set $\mathcal{S} = \{s_0, \cdots, s_{52}\}$ of all odd primes
             lower than 256 (stored in ROM), a number $t$ of Miller-Rabin tests to perform
   Output: A probable prime $p$

   /* Generate a seed                                                        */
 1 Randomly generate an odd $\ell$-bit integer $v_0$
   /* Prime Sieve                                                            */
 2 $v \leftarrow v_0$
 3 $s \leftarrow s_0$
 4 $i = 0$
 5 while $(v \bmod s \neq 0)$ and $(i < 53)$ do
 6     $i = i + 1$
 7     $s \leftarrow s_i$
 8 if $(i \neq 53)$ then
 9     $v = v + \tau$
10     goto Step 3
   /* Probabilistic primality tests                                          */
11 else
12     $i = 0$
       /* Process $t$ Miller-Rabin's tests (stop if one fails)               */
13     while $(\texttt{Miller-Rabin}(v) = ok)$ and $(i < t)$ do
14         $i = i + 1$
   /* Process 1 Lucas' test[a]                                               */
15 if $(i = t)$ and $(\texttt{Lucas}(v) = ok)$ then
16     return $v$
17 else
18     $v = v + \tau$
19     goto Step 3
```

[a] Miller-Rabin's tests are followed by one Lucas' test because there is no known composite integer n for which they are both reporting that n is probably prime.

2.2 Algorithm's Improvement: An Up-to-Date Version

In practice, implementations of Algorithm 1 are often improved further by exploiting the fact that the sieve elements s_j are very small compared to the prime candidate v. The idea, mentioned in ANSI X9.31 [1] and by Brandt *et al.* in [6], is to replace costly modular reductions over ℓ-bit integers by fast reductions over 8-bit integers[3]. Indeed, by construction, the reduction $v \bmod s$ at Step 5 for the $(i+1)^{\text{th}}$ prime candidate may indeed be rewritten as $v_0 + i \cdot \tau \bmod s_j$, for s_j a prime in the sieve. Written differently, this relation can also be expressed as $(v_0 \bmod s_j) + i \cdot \tau \bmod s_j$. As a consequence, one can start by computing all the remainders $r_{0j} = (v_0 \bmod s_j)$ and by storing them in a RAM table R (containing 53 bytes). Then, the prime sieve for the next candidate v_1 is simply done by updating R such that $R[j] = R[j] + \tau \bmod s_j$ for any $j < 53$. After this step, which only processes 8-bit values as long as τ is small enough[4] , R contains all

[3] The choice of 8-bit integers here comes from an efficiency argument and is not related to the architecture of the device (see Section 2.1).

[4] If $\tau = 2$, since the greatest sieve element in our implementation is strictly lower than $256 - 2$, the value $R[j] + \tau$ can always be stored in a byte.

the remainders $r_{1j} = v_1 \mod s_j$. More generally, this idea can be applied recursively to efficiently deduce the remainders related to the candidate v_i from those related to the previous one v_{i-1}. Eventually, after each update of R, the result of the prime sieve for a candidate is obtained by checking whether R contains a null remainder or not.

The efficiency improvement described above leads to replace Steps 2-10 in Algorithm 1 (before the probabilistic tests) by the ones provided by Algorithm 2.

Algorithm 2. Improved Prime Sieve

```
   /* Prime Sieve for $v_0$                                                    */
1  for $j = 0$ to 52 do
2      $R[j] \leftarrow v_0 \mod s_j$        /* costly modular reduction over $\ell$-bit integers */
3
   /* Prime Sieve for $v_i$ with $i > 0$                                       */
4  $v \leftarrow v_0$
5  while (R contains a null remainder) do
6      $v = v + \tau$
7      for $j = 0$ to 52 do
8          $R[j] \leftarrow R[j] + \tau \mod s_j$     /* efficient modular reduction over 8-bit integers */
9
```

Remark 1. Usually, reductions at Step 2 of Algorithm 2 are performed by calling the arithmetic coprocessor whereas those at Step 8 are done with standard `CPU` instructions[4]. For instance, in a `8051` architecture the instruction `DIV` may be used to compute the remainder.

In addition to its efficiency, Algorithm 2 has a side advantage: the prime sieve is regular[5] which renders Finke *et al.* 's attack [16] ineffective. The gain in efficiency and in security explains why an up-to-date implementation of Algorithm 1 must involve the improved prime sieve described in Algorithm 2. For this reason, our attack in the next section is described against such an implementation. It must however be mentioned that it can also be applied against a straightforward implementation of Algorithm 1, in addition to Finke *et al.* 's attack.

3 A New Attack

3.1 Core Idea

The attack developed in this section aims at recovering information on a probable prime p generated by Algorithm 1, implemented with the improvements described in Algorithm 2. For this purpose, let us focus on this algorithm when the prime sieve is applied to test whether the $(i+1)^{\text{th}}$ candidate v_i has small factors. During this process, the following remainders are computed for every s in the sieve set $\mathcal{S}$:

$$r_i = v_i \mod s \ . \tag{1}$$

[5] Assuming that testing whether the elements of R are non-zero is done with caution.

Knowing that v_i has been generated deterministically from a seed v_0 by an iterative increment of τ, Equation (1) can be rewritten as: $r_i = v_0 + i \cdot \tau \mod s$. Moreover, if n denotes the number of tested candidates, the probabilistic prime p returned by Algorithm 1 satisfies the following equation: $p = v_0 + (n-1)\tau$. Eventually combining the two previous relations shows that the secret prime p and the remainders r_i are linked through the following equation:

$$r_i \equiv p - (n - i - 1) \cdot \tau \pmod{s} \ . \tag{2}$$

When the value n is made public, the remainder r_i is a function of both the secret p and a known value $(n-i-1)\tau$ (recalling that τ is public as part of the algorithm specification). From that point, if we denote by ℓ_i the measured device activity (*e.g.* power consumption or electromagnetic emanations) coming from the manipulation of r_i, then an SCA can straightforwardly be defined assuming that an attacker is able to isolate the trace ℓ_i for all $i < n$. Indeed, the sample $\{\ell_i; i < n\}$ can be compared with the predictions deduced from both the values $\{(n-i-1) \cdot \tau; i < n\}$ and an hypothesis on $p \bmod s$. This type of SCA, where a single algorithm execution is observed, is called *horizontal* in [2,9]. When n is large enough, this attack leads to the recovery of p modulo s (*i.e.* brings $\log_2(s)$ bits of information on p).

Eventually, the attack is applied for every prime s in the set sieve $\mathcal{S}$ and all results $p \bmod s$ are combined through the Chinese Remainder Theorem to reconstruct p modulo $\prod_{s \in \mathcal{S}} s$. This leads to the recovery of $\log_2(\prod_{s \in \mathcal{S}} s)$ bits of information on p. Of course, this situation corresponds to a perfect attack scenario where each SCA against $p \mod s$ succeeds. In practice, some of them will likely fail, which reduces the amount of recovered information.

The practical soundness of the assumption that n is known by the adversary and that he/she is able to isolate the leakage traces ℓ_i (which are prerequisites for our attack to be applicable) is studied in Section 4.1.

3.2 Full Description

In this section, we denote by r_{ij} the remainder corresponding to the division of the $(i+1)^{\text{th}}$ candidate v_i by the $(j+1)^{\text{th}}$ sieve element s_j. Moreover, we use the notation ℓ_{ij} to refer to the measured device activity[6] during the processing of r_{ij}. Once all the measurements have been obtained, the adversary splits them into different samples $(\ell_{ij})_i$, one for each sieve element s_j. Each sample can thus be viewed as a set of noisy observations of the remainders $(r_{ij})_i$ satisfying (2) for $s = s_j$. Assuming that the prime generation algorithm outputs the n^{th} tested candidate[7], then the size of each sample $(\ell_{ij})_i$ is n. To sum up, we have the following relation:

[6] Each ℓ_{ij} can be viewed as a vector of real values whose size depends on the sampling rate of the oscilloscope used for the measurements and the manipulation time of r_{ij} by the device.

[7] Which means that the candidate v_{n-1} is the first that has successfully passed all the primality tests

$$\ell_{ij} \hookleftarrow r_{ij} = p - (n - i - 1) \cdot \tau \bmod s_j \ , \tag{3}$$

where $\hookleftarrow$ denotes a noisy observation. With these different samples $(\ell_{ij})_i$ in hand, the adversary is now able to target each sieve element independently. Namely, for each j, the adversary will try to recover $p \bmod s_j$ by exhaustively testing all possible values that can be reached by this expression[8]. The test of each hypothesis, say h, on $p \bmod s_j$ is simply done by following the classical outlines of an SCA attack:

- use a leakage model $\mathbf{m}$ to deduce a set of predictions $\{\mathbf{m}(h - (n - i - 1)\tau \bmod s_j); i < n\}$. A possible choice for $\mathbf{m}$ is the Hamming weight function HW (as done in Section 3.3) but, if needed, more accurate models can be built by performing analyses based on *Linear Regression* [15,24];
- apply an SCA distinguisher Δ (*e.g.* a correlation coefficient) to compare the predictions with the measurements and to validate or invalidate the hypothesis.

In other words, a classical horizontal SCA as in [2,9,10] is performed against each secret $(p \bmod s_j)$, using the fact that this value is manipulated several times, combined with a known value of the form $(n - i - 1)\tau \bmod s_j$ with $i < n$. Each such attack, that will be called partial in the sequel, outputs a most likely candidate for $(p \bmod s_j)$. In case of success, it brings $\log_2(s_j)$ bits of information on p. We sum-up in Algorithm 3 the different steps of the full attack. The size of the sieve set $\mathcal{S}$ is denoted by λ (we remind that it equals 53 in the standard implementation detailed in Section 2).

Algorithm 3. Attack Against Prime Generation Algorithm

```
/* Measurements Phase                                              */
for $i = 0$ to $n - 1$ do
    for $j = 0$ to $\lambda - 1$ do
        measure $\ell_{ij}$

/* Attack Phase:                                                   */
/* for each sieve, perform a partial SCA                           */
for $j = 0$ to $\lambda - 1$ do
    /* ... test each possible candidate ...                        */
    for $h = 1$ to $s_j - 1$ do
        /* ... by processing predictions ...                       */
        for $i = 0$ to $n - 1$ do
            $m_{ij} = \mathbf{m}(h - (n - i - 1)\tau \bmod s_j)$
        /* ... and applying a statistical distinguisher ...        */
        score[$h$] = $\Delta((m_{ij})_i, (\ell_{ij})_i)$
    /* ... then select the most likely candidate ...               */
    candidate[$j$] = $\mathtt{argmax}_h$(score[$h$])

/* Apply the Chinese Remainder Theorem (CRT)                       */
$\hat{p}$ = CRT (candidate[0] mod $s_0, \cdots$, candidate[$\lambda - 1$] mod $s_{\lambda-1}$)
return $\hat{p}$
```

[8] Which excludes 0 since p is prime.

Note that the attack described in Algorithm 3 could also be adapted to target straightforward implementations of Algorithm 1. The only difference is that the adversary will not have the same number of observations for each sieve element. Indeed, as the prime sieve is stopped each time a divisor is found, the probability that r_{ij} is processed (and thus observed) decreases with respect to j. As we think that such a straightforward implementation of Algorithm 1 is unlikely to be implemented in secure devices (because it is not efficient and vulnerable to Finke *et al.* 's attack – see Section 2–), we decided not to detail it in this paper.

3.3 Attack Analysis

In this section we first study, for typical bit-lengths $\ell \in \{256, 512, 1024\}$, the number n of prime sieve processings that can be observed by an attacker during the generation of a probable prime of size ℓ. Then, we focus on the success rate of the attack (*i.e.* its ability to completely recover p) under different hypotheses on n. For simplicity and because this is a common choice in practice, we choose to focus on the case $\tau = 2$.

About the number of prime sieve processings. The effectiveness of our attack strongly depends on the number n of leakage values that can be retrieved for each sieve element. This value, which is also the number of prime sieve processings, depends on the seed v_0; thus, contrary to what happens in classical SCA, it cannot be *a priori* chosen by the adversary[9].

On Figure 1, several estimations of the *complementary cumulative distribution function* (ccdf) $\overline{F}_n(x)$ of n, viewed as a random variable, are plotted. Namely, each curve corresponds to an estimation[10] of the probability $\overline{F}_n(x)$ (in ordinate) that n is greater than or equal to some value x (in abscissa). The three plotted curves correspond to prime generations for a bit-length ℓ equal to 256, 512 and 1024 respectively. In the sequel, we focus on the 512-bit case (even if the outlines of our approach could also be followed to study the two other cases) since generating primes of that size is for instance required when constructing a 1024-bit RSA modulus (*e.g.* for some banking applications) or when generating strong primes according to the ANSI X9.31 standard [1]. For a 512-bit prime, the *median* of the distribution of n as well as the first and third *complementary quartiles*[11], are respectively equal to 53, 126 and 246. The quartiles Q_1, Q_2 and Q_3 related to $75\%, 50\%$ and 25% are represented by horizontal lines in Figure 1.

Attack Effectiveness. Let us now focus on the ability of our attack to recover x bits of information on p by combining the results of the partial CPA attacks

[9] In classical SCA, the number of observations is chosen and increased until the attack achieves some success rate.

[10] Estimations have been done over 2000 observations of n, namely for 2000 prime generations.

[11] We recall that the median of a random variable X is the value Q_2 such that $\Pr(X \leq Q_2) = 0.5$. Similarly, the first (resp. the third) complementary quartile of X is the value Q_1 (resp. Q_3) s.t. $\Pr(X > Q_1) = 0.75$ (resp. $\Pr(X > Q_3) = 0.25$).

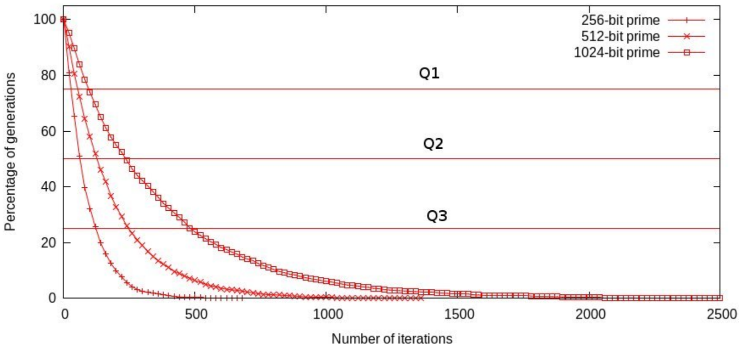


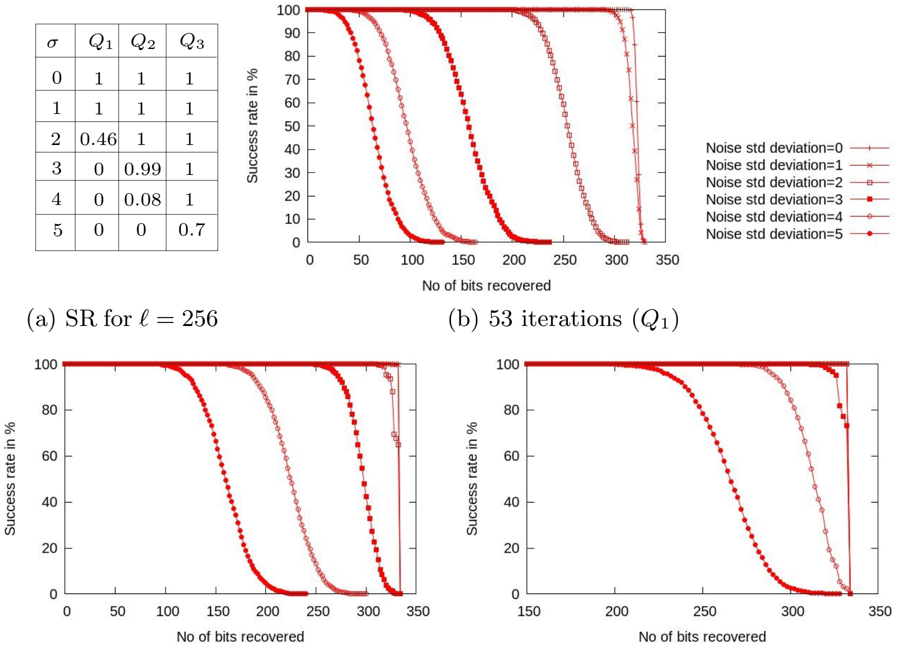
Fig. 1. Cumulative distribution function of n for different prime bit-lengths ℓ

against the remainders $p \bmod s_j$. We here assume that the attacker is able to detect when a partial CPA returns a correct result. It may first be noticed that a correct guess on all these remainders provides 333 bits of information on p (assuming that $\mathcal{S}$ contains the 53 smallest primes). As argued in the next paragraph, this upper bound[12] limits the size of the prime numbers which can be successfully recovered with our attack. In Figure 2, we plot the probability (in ordinate) that our attack recovers at least x bits of information on p. As done for the previous figure, the probabilities have been computed from simulations in three different contexts depending on whether the number n of leakage values per CPA equals the first complementary quartile $Q_1 = 53$, the median $Q_2 = 126$ or the third complementary quartile $Q_3 = 246$. Several results are moreover presented, corresponding to different amounts of noise in the observations. For each quartile, the success rates have been estimated with 2000 attacks.

Before analysing the simulation results in Figure 2, it remains to define when our attack is considered to succeed. For this purpose, we recall that the generated prime p is assumed to be afterwards used to define an RSA modulus. In such a context, a well-known technique introduced by Coppersmith [13,14] may be applied to reconstruct p from approximately half of its bits[13]. This technique works by translating the problem of recovering the unknown part of p into that of finding a small root on a bivariate polynomial equation. Such an issue can then be solved by performing a lattice reduction on a well-chosen basis. In our context, the number of bits that have to be retrieved to lead to the full recovery of a prime p with bit-length 512 is $256 = 512/2$. We are aware that this bound is theoretical since it can only be achieved when reducing a lattice of infinite

[12] Since the upper bound increases with the number of primes involved in the sieving, the same holds for the size of the probable primes concerned by our attack.

[13] To be more accurate, Coppersmith's original technique aims at recovering p knowing the half most (or least) significant bits. In our context, one gets a relation of the form $p \equiv p_0 \bmod \prod_{s_j \in \mathcal{S}} s_j$ with a known value p_0. This case can be handled using a slight generalisation of the original method, under the condition that $\prod_{s_j \in \mathcal{S}} s_j$ is approximately half the bit length of p (see Corollary 2.2 in [3]).



σ	Q_1	Q_2	Q_3
0	1	1	1
1	1	1	1
2	0.46	1	1
3	0	0.99	1
4	0	0.08	1
5	0	0	0.7

(a) SR for $\ell = 256$

(b) 53 iterations (Q_1)

(c) 126 iterations (Q_2)

(d) 246 iterations (Q_3)

Fig. 2. Number of bits retrieved from different noise levels

dimension. In practice, several additional bits are required for Coppersmith's method to work[14]. Nevertheless, the problem can be circumvented (even if the exact bound of "256 bits" can never be achieved in practice) at the price of an exhaustive search on the missing values, thus making the overall complexity of the attack increase. In our case, since we use Coppersmith's method as a black-box, we choose to define a successful attack as recovering 256 bits on p (this bound thus corresponds to an "ideal" scenario) .

The results are summed up in Figure 2(a). Not surprisingly, the attack works better and recovers more bits on p when the number of tested candidates n increases (we indeed have more observations to recover each sensitive remainder). In the case where there is a lot of noise or few iterations, the expected number of bits correctly guessed on p drops. These results can be exploited to obtain a lower bound on the overall success rate (SR for short) of our attack:

- [For $\sigma = 1$]: the attack recovers 256 bits of information on p with probability 1 for $n = Q_1$, Q_2 and Q_3. In other terms, our attack succeeds for all the prime generations where n reaches the first quartile, that is for 75% of the generations. We can thus estimate the lower bound for our success probability in this case by $p_{\sigma=1} \geq 0.75$.

[14] See [16] for heuristic results with respect to various numbers of retrieved bits.

- [For $\sigma = 2$]: the attack recovers 256 bits of information on p with probability 1 for $n = Q_2$ and Q_3 and with probability 0.46 for $n = Q_1$. We can thus estimate the lower bound for our success probability in this case by $p_{\sigma=2} \geq (0.46 + 1 + 1)/4 = 0.615$.
- [For $\sigma \geq 3$]: we can estimate similarly the lower bound for our success probability in the remaining cases by $p_{\sigma=3} \geq 0.4975$, $p_{\sigma=4} \geq 0.27$ and $p_{\sigma=5} \geq 0.175$ respectively.

4 Attack Flow in Practice

4.1 Discussion on the Measurements Phase

In this section, we come back to the attack hypotheses made in Section 3 and we argue about their relevance. Namely, we study the practical soundness of the assumption that the number of tested candidates n is known by the adversary and that he/she is able to isolate the leakage traces ℓ_{ij} defined as in Equation (3). For this purpose, we consider here an implementation of a 512-bit prime number generation, computed on a smart-card micro-controller equipped with an 8-bit CPU and a modular arithmetic co-processor, both running at several dozens of MHz. This implementation corresponds to an off-the-shelf smart-card. To simplify the analysis, we directly focus on the case where the attack described in Section 3 is effective with high probability. For this reason, we developed our argumentation under the hypothesis that the number of tested candidates n is at least 250 (which happens with probability 25% – see Figure 1 –).

Let us now evaluate the time required by the platform to process a prime number generation as specified in the previous paragraph. Thanks to the sieving pre-processing, a probabilistic primality test (here a sequence of Miller-Rabin tests) is performed for 1 candidate over 10 in average (see Mertens' theorem [25]). Let t be the maximum number of Miller-Rabin tests that must be passed by a candidate. Observing that each test takes 10ms on the considered platform, then the full processing time of the algorithm is upper bounded by $250t$ms. For instance, when $t = 10$, which is a reasonable value to ensure the primality of a number with satisfying probability, the full processing time is 2.5s. Note that this approximation does not take into account the time spent in the 250 efficient prime sieves, since this is negligible in comparison to the rest of the algorithm (see Section 2.2). For this (practical) attack scenario, several issues arise during the measurements phase (where we denote by i the number of tested candidates):

1. how to record the long side-channel trace corresponding to the full prime number generation computation, or at least to the i efficient prime sieve tests[15];
2. how to recognize and extract the patterns corresponding to the i prime sieve tests and how to convert them into i smaller side-channel traces;
3. in each small side-channel trace previously created, how to precisely align the sub-patterns corresponding to the trial divisions (Step 8 in Algorithm 2).

[15] Meaning i iterations of the while loop in Algorithm 2.

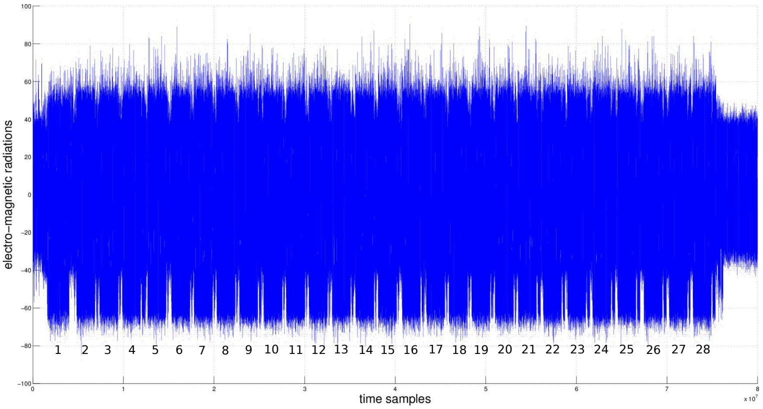


Fig. 3. Electro-magnetic radiations measured during a prime number generation computation on a commercial smartcard. Pattern 1 corresponds to the initial costly prime sieve, whereas patterns 2 to 28 correspond to Miller-Rabin tests.

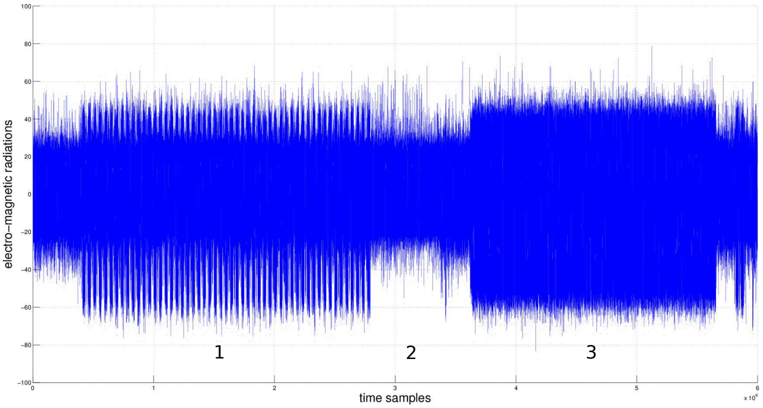


Fig. 4. Zoom on the two first patterns of Figure 3. Pattern 1 corresponds to the initial costly prime sieve, whereas pattern 3 corresponds to the first Miller-Rabin test. First efficient prime sieves (with small integer divisions) are located inside pattern 2.

Solving the first issue depends on the specifications of the oscilloscope used to record the long side-channel trace. More precisely, it depends on its channel memory depth, *i.e.* the number of samples the oscilloscope can record per channel during one single acquisition. To record 250 iterations with a sampling rate of 100MSamples per second (which is a minimum on such platforms to perform a CPA), then the channel memory depth must be at least of 250MSamples, which is available on high-end oscilloscopes. The oscilloscope trigger can moreover be set-up to skip the recording of the first prime sieve computation (Step 1-3 of Algorithm 2), as it is not used in our attack. This amounts to skip the step corresponding to the pattern 1 in Figures 3 and 4.

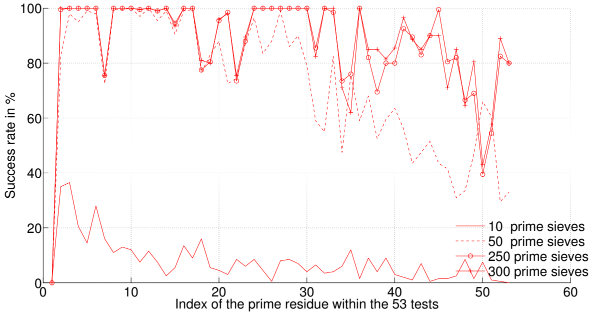
Once the long side-channel trace has been acquired, the second issue consists in recognizing patterns corresponding to the efficient prime sieve computations. Such patterns are located between those corresponding to probabilistic primality tests, which have a particular side-channel signature due to the use of the modular arithmetic co-processor. The Miller-Rabin tests correspond to patterns 2 to 28 on Figure 3, and to pattern 3 on Figure 4. Thanks to this patterns identification phase, one can then deduce that several prime sieve computations are located inside Pattern 2 of Figure 4. Once such patterns have been found, classical automated pattern matching techniques can eventually be used to extract the other ones in the rest of the long side-channel trace.

Finally, the third issue should be solved thanks to peak extraction techniques classically used in SCA. This would enable to align the patterns corresponding to the trial divisions in each small side-channel trace. On the traces we acquired (Figures 3 and 4), the signal is too noisy for such alignment. In the following we continue our practical analysis on a toy implementation of the prime sieve running on a different architecture than that used in this sub-section.

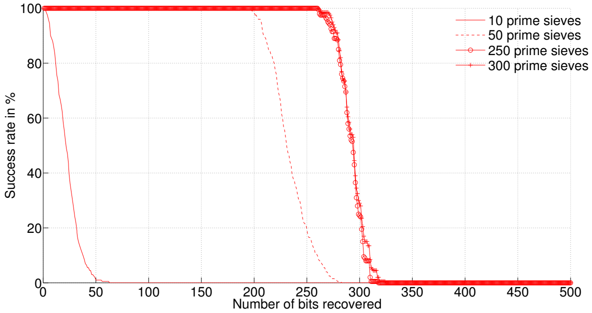
4.2 Experiments on a Toy Implementation

To confirm the analyses conducted in Section 3.3 and to validate our assumptions in practice, the new attack has been tested against a toy implementation embedded on an 8-bit ATMega128 micro-controller running at 8MHz. For simplicity reasons, we did not implement the full probabilistic prime generation described in Algorithm 1 but only 300 iterations of the loop corresponding to steps 5-9 in Algorithm 2 parametrised with a random seed v_0. As our attack only targets the prime sieve and not the probabilistic tests, this choice does not impact the soundness of the conclusions we are going to draw from the experimentations reported below.

The electro-magnetic activity of the device during the processing of the 300 prime sieve tests has been measured with a sampling rate of 1GSamples per second. 300×53 patterns have then been extracted. These patterns correspond to the trial divisions of the 300 prime candidates $v_i = v_0 + 2i$ by the 53 prime sieve elements s_j (Steps 5-9 in Algorithm 2). Afterwards, the attack described in Algorithm 3 has been performed with the Pearson correlation coefficient as statistical distinguisher Δ. The overall experiment (including the acquisition phase) has been repeated 200 times. Following the same approach as in Section 3.3, the effectiveness of our attack has then been studied under the assumption that the targeted prime value p was known for each experiment. This assumption makes it possible to decide whether each partial attack on $p \bmod s_j$ succeeded or not, and hence allowed us to apply the Chinese Remainder Theorem only with the correct guesses. The results are reported in Figures 5(a) and 5(b). They correspond to attack scenarios where the number of exploited prime sieve observations (among the 300 ones) was respectively limited to 10, 50, 250 and 300. Figure 5(b) must be viewed as the experimental equivalent of the simulations described in Figure 2.



(a) Success rates for each prime sieve elements (over 200 attack experiments)



(b) Success rates for recovering x bits of information on the generated prime

Fig. 5. Success rates in practice

Even if the experimental success rates are slightly below those obtained for our attack simulations[16] (see Section 3.3 for a theoretical analysis), the general trends are the same. In particular, our attack succeeds in recovering more than 256 bits of information with success probability 0.9 as long as the number n of observed prime sieve tests is at least equal to 250 (which happens with probability 0.25 when the prime length ℓ equals 512, see Section 3.3). This not only confirms the soundness of the analysis in Section 3.3 but also demonstrates the practicality of our attack.

Let us now focus on a real attack context where the assumption "the target prime p is known" has been relaxed. In this scenario, the adversary loses his ability to decide for each partial SCA (against the remainder of p modulo a prime sieve element) whether it has succeeded or not. Consequently, he cannot select which remainders to keep for the recombining phase and must hence apply the

[16] This can easily be explained by the higher noise encountered during the practical experiments and the fact that the Hamming weight leakage model used in the CPA does not perfectly fit the real leakage function.

Chinese Remainder Theorem (CRT) on all the partial SCA results (as described in Step 10 of Algorithm 3). This attack will thus only work if all the retrieved remainders are correct, which occurs with a probability that can be approximated by the product of the 53 success rates plotted in Figure 5(a). Even for $n = 300$, it can be checked that this probability is very small. Fortunately, several strategies can be applied to significantly improve this success rate.

4.3 Avenues of Improvement

Larger primes generation. Our attack would not work for primes beyond 666 bits, since the 53 prime sieve elements s_j only permit to retrieve a maximum of 333 bits on p. However the analysis can easily be adapted to 1024-bit primes, when the prime generation algorithm uses a larger sieve set $\mathcal{S}$ (requiring a product of its elements larger than 1024 bits).

Case of RSA modulus. For RSA key generation, the adversary may attack the two prime factors p and q independently. Then, the public relation $N = pq$ can be used to compare the remainder hypotheses returned by the partial SCA of each attack. Such a procedure is thoroughly described in [16]. The attacker can also gain some information about the secret exponent d through the analysis of the equation $e \cdot d = k(N - (p+q) + 1) + 1$. When e is small, implying k small too, one can deduce information about $d \bmod s_j$, knowing $p \bmod s_j$ and $q \bmod s_j$.

Key Enumeration Approach. Instead of selecting only the remainder that maximizes the distinguisher value (as presented in Step 9 of Algorithm 3), one could choose to record the scores associated to any remainder hypothesis h for any prime sieve element s_j. Then, instead of applying the CRT recombining to only one 53-tuple (as in Step 10 of Algorithm 3), we can do it for all the 53-tuples of hypotheses from the most to the least likely, until the correct p is recovered (after applying Coppersmith technique, it should factor the RSA public modulus). A straightforward application of this strategy is clearly inefficient if the correct guess is not reconstructed after few steps. To optimise this phase, it is recommended to use a so-called *key-enumeration algorithm* (KEA) (see Appendix A for an efficient algorithm proposed by Veyrat-Charvillon *et al.* [26]).

Initial Prime Sieve. Additional information may be retrieved during the initial *expensive* prime sieve. Such information however is likely to be very different (in nature) than the information retrieved by the following sieves (since the operations are probably handled by different part of the hardware) and then should not be used directly during the CPA attack.

5 Conclusion and Countermeasures Proposal

In this paper, we have described an attack against prime number generation. Compared to the existing attack of [28], this attack defeats a protected implementation of the probable prime tests with a regular prime sieve. Our attack exploits two features of a prime generation algorithm: the use of a prime sieve and a deterministic candidates generation. Such algorithms are for example described in the well-known norms ANSI X9.31 and FIPS 186-4 [1,17]. We gave an analysis of the efficiency of our attack and demonstrated its practicality against a smart-card toy implementation (which confirms our analyses).

Several approaches can be followed to thwart our attack. A first one is to randomly add dummy trial divisions in each prime sieve computation. Another one is to perform each prime sieve computation in a pseudo-random order. Both countermeasures have the effect to misalign trial divisions, and then to increase the noise in the measurements. A different approach would be to choose a prime generation algorithm without the two features required in our attack. For example, Fouque and Tibouchi [18] propose a prime generation with a non-deterministic generation of prime candidates. Another recent proposal is the efficient provable prime generation algorithm of Clavier *et al.* [11].

Acknowledgement. Aurélie Bauer's research was supported in part by the French ANR-12-JS02-0004 ROMAnTIC Project.

References

1. ANSI X9.31. Digital Signature Using Reversible Public Key Cryptography for the Financial Services Industry. American National Standards Institute (1998)
2. Bauer, A., Jaulmes, E., Prouff, E., Wild, J.: Horizontal and Vertical Side-Channel Attacks against Secure RSA Implementations. In: Dawson, E. (ed.) CT-RSA 2013. LNCS, vol. 7779, pp. 1–17. Springer, Heidelberg (2013)
3. Boneh, D., Durfee, G., Frankel, Y.: An Attack on RSA Given a Small Fraction of the Private Key Bits. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 25–34. Springer, Heidelberg (1998)
4. Boscher, A., Naciri, R., Prouff, E.: CRT RSA Algorithm Protected Against Fault Attacks. In: Sauveron, D., Markantonakis, K., Bilas, A., Quisquater, J.-J. (eds.) WISTP 2007. LNCS, vol. 4462, pp. 229–243. Springer, Heidelberg (2007)
5. Brandt, J., Damgård, I.B.: On Generation of Probable Primes by Incremental Search. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 358–370. Springer, Heidelberg (1993)
6. Brandt, J., Damgård, I., Landrock, P.: Speeding Up Prime Number Generation. In: Imai, H., Rivest, R.L., Matsumoto, T. (eds.) ASIACRYPT 1991. LNCS, vol. 739, pp. 440–449. Springer, Heidelberg (1993)
7. Chevallier-Mames, B., Ciet, M., Joye, M.: Low-cost Solutions for Preventing Simple Side-Channel Analysis: Side-Channel Atomicity. IEEE Transactions on Computers 53(6), 760–768 (2004)
8. Clavier, C., Coron, J.-S.: On the Implementation of a Fast Prime Generation Algorithm. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 443–449. Springer, Heidelberg (2007)
9. Clavier, C., Feix, B., Gagnerot, G., Giraud, C., Roussellet, M., Verneuil, V.: ROSETTA for Single Trace Analysis – Recovery of Secret Exponent by Triangular Trace Analysis. In: Galbraith, S., Nandi, M. (eds.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 140–155. Springer, Heidelberg (2012)

10. Clavier, C., Feix, B., Gagnerot, G., Roussellet, M., Verneuil, V.: Horizontal Correlation Analysis on Exponentiation. In: Soriano, M., Qing, S., López, J. (eds.) ICICS 2010. LNCS, vol. 6476, pp. 46–61. Springer, Heidelberg (2010)
11. Clavier, C., Feix, B., Thierry, L., Paillier, P.: Generating Provable Primes Efficiently on Embedded Devices. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 372–389. Springer, Heidelberg (2012)
12. Clavier, C., Joye, M.: Universal Exponentiation Algorithm – A First Step towards Provable SPA-Resistance. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 300–308. Springer, Heidelberg (2001)
13. Coppersmith, D.: Finding a Small Root of a Bivariate Integer Equation; Factoring with High Bits Known. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 178–189. Springer, Heidelberg (1996)
14. Coppersmith, D.: Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities. Journal of Cryptology 10(4), 233–260 (1997)
15. Doget, J., Prouff, E., Rivain, M., Standaert, F.-X.: Univariate Side Channel Attacks and Leakage Modeling. Journal of Cryptographic Engineering 1(2), 123–144 (2011)
16. Finke, T., Gebhardt, M., Schindler, W.: A New Side-Channel Attack on RSA Prime Generation. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 141–155. Springer, Heidelberg (2009)
17. FIPS PUB 186-4. Digital Signature Standard (DSS). Federal Information Processing Standards Publication (July 2013)
18. Fouque, P.-A., Tibouchi, M.: Close to Uniform Prime Number Generation With Fewer Random Bits. IACR Cryptology ePrint Archive, 2011:481 (2011)
19. Gérard, B., Standaert, F.-X.: Unified and Optimized Linear Collision Attacks and Their Application in a Non-profiled Setting. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 175–192. Springer, Heidelberg (2012)
20. Giraud, C.: An RSA Implementation Resistant to Fault Attacks and to Simple Power Analysis. IEEE Transactions on Computers 55(9), 1116–1120 (2006)
21. Gordon, J.: Strong Primes are Easy to Find. In: Beth, T., Cot, N., Ingemarsson, I. (eds.) EUROCRYPT 1984. LNCS, vol. 209, pp. 216–223. Springer, Heidelberg (1985)
22. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press (1997)
23. Moreno, C., Hasan, M.A.: SPA-Resistant Binary Exponentiation with Optimal Execution Time. Journal of Cryptographic Engineering 1(2), 87–99 (2011)
24. Schindler, W.: Advanced Stochastic Methods in Side Channel Analysis on Block Ciphers in the Presence of Masking. Journal of Mathematical Cryptology 2, 291–310 (2008)
25. Forschungszentrum Graz. Mathematisch-Statistische Sektion. Berichte Der Mathematisch-Statistischen Sektion Im Forschungszentrum Graz (1973)
26. Veyrat-Charvillon, N., Gérard, B., Renauld, M., Standaert, F.-X.: An Optimal Key Enumeration Algorithm and Its Application to Side-Channel Attacks. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 390–406. Springer, Heidelberg (2013)
27. Vigilant, D.: RSA with CRT: A New Cost-Effective Solution to Thwart Fault Attacks. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 130–145. Springer, Heidelberg (2008)
28. Vuillaume, C., Endo, T., Wooderson, P.: RSA Key Generation: New Attacks. In: Schindler, W., Huss, S.A. (eds.) COSADE 2012. LNCS, vol. 7275, pp. 105–119. Springer, Heidelberg (2012)

A Key Enumeration Algorithm

The idea developed by the authors of [26] is to produce, one after another, the 16-byte hypotheses on the AES master key. The 8-bit sub-keys of each hypothesis are returned independently from 16 different SCA attacks and then concatenated together in order to be tested as the cipher secret key. This set-up is in fact very similar to ours: instead of 16 bytes, we consider 53 independent secret of different lengths. From these secret hypotheses, a part of the secret prime is recovered through CRT recombining and then used to recover the whole secret prime. Similarly to the work of Gérard and Standaert in [19], a Bayesian extension can be computed over the correlation coefficient values for each of the 53 independent attacks. Hence, to each small prime s_j, and each remainder hypothesis h in the set $(\mathbb{Z}/s_j\mathbb{Z})^\star$ is associated the following probability $\Pr[h = p \bmod s_j \mid \{\ell_{ij}\}_i]$, where the set of consumption traces $\{\ell_{ij}\}_{i,j}$ is defined as in Equation (3).

Once the latter probability has been computed for any value h and any s_j, the recursive algorithm proposed in [26] can be straightforwardly applied to provide the list of 53-tuples of remainder hypotheses ordered from the most to the less likely hypothesis. We do not recall the algorithm here (a detailed description can be found in [26]).

Further Improvements. For the complete attack to be successful (*e.g.* factoring an RSA modulus), it is not necessary to recover all the 53 remainders of p but only a sufficient number of them s.t. their product gives 256 bits of information (instead of the 333 bits given by the product of all the 53 first small primes). In view of this, the attacker goal is no longer to recover all the remainders $p \bmod s_j$ such that s_j in $\mathcal{S}$ but a subset of them which brings 256 bits of information. Let us denote by $\{\mathcal{S}_1, ..., \mathcal{S}_m\}$ a family of m subsets satisfying the latter property. The KEA algorithm recalled previously can now be applied to each subset independently (taking into account the corresponding CPA attacks). The brute-force processing then takes simultaneously the m sets of attack results and, at each step, the most likely hypothesis is chosen among the most likely hypothesis of each set. The respective KEA instance is afterwards advanced to the next best solution. Such multi-set approach would definitely improve the attack efficiency.