A New Side-Channel Attack on RSA Prime Generation

Thomas Finke, Max Gebhardt, Werner Schindler

Bundesamt für Sicherheit in der Informationstechnik (BSI)
Godesberger Allee 185–189
53175 Bonn, Germany
{Thomas.Finke,Maximilian.Gebhardt,Werner.Schindler}@bsi.bund.de

Abstract. We introduce and analyze a side-channel attack on a straight-forward implementation of the RSA key generation step. The attack exploits power information that allows to determine the number of the trial divisions for each prime candidate. Practical experiments are conducted, and countermeasures are proposed. For realistic parameters the success probability of our attack is in the order of 10–15 %.

Keywords: Side-channel attack, RSA prime generation, key generation.

1 Introduction

Side-channel attacks on RSA implementations have a long tradition (e.g. [8, 9, 12, 13]). These attacks aim at the RSA exponentiation with the private key d (digital signature, key exchange etc.). On the other hand only a few papers on side-channel attacks, resp. on side-channel resistant implementations, exist that focus on the generation of the primes p and q, the private key d and the public key $(e, n = pq)$ (cf., e.g. [4, 1]). If the key generation process is performed in a secure environment (e.g., as part of the smart card personalisation) it is infeasible to mount any side-channel attack. However, devices may generate an RSA key pair before the computation of the first signature or when applying for a certificate. In these scenarios the primes may be generated in insecure environments.

Compared to side-channel attacks on RSA exponentiation with the secret key d the situation for a potential attacker seems to be less comfortable since the primes, resp. the key pair, are generated only once. Moreover, the generation process does not use any (known or chosen) external input.

We introduce and analyse a power attack on a straight-forward implementation of the prime generation step where the prime candidates are iteratively incremented by 2. The primality of each prime candidate v is checked by trial divisions with small primes until v is shown to be composite or it has passed all trial divisions. To the 'surviving' prime candidates the Miller-Rabin primality test is applied several times. We assume that the power information discovers the number of trial divisions for each prime candidate, which yields information on p and q, namely $p (\bmod s)$ and $q (\bmod s)$ for some modulus s, which is a product of small primes. The attack will be successful if s is sufficiently large. Simulations and experimental results show that for realistic parameters (number of small primes for the trial divisions under consideration of the magnitude of the RSA primes) the success probability is in the order of 10–15%, and that our assumptions on the side-channel leakage are realistic.

Reference [4] considers a (theoretical) side channel attack on a careless implementation of a special case of a prime generation algorithm proposed in [7] that is successful in about 0.1% of the trials. Reference [3] applies techniques from [1], which were originally designated for the shared generation of RSA keys. We will briefly discuss these aspects in Section 6.

The intention of this paper is two-fold. First of all it presents a side-channel attack which gets by with weak assumptions on the implementation. Secondly, the authors want to sensibilise the community that RSA key generation in potentially insecure environments may bear risks. The authors want to encourage the community to spend more attention on the side-channel analysis of the RSA key generation process.

The paper is organized as follows: In Section 2 we have a closer look at the RSA prime generation step. In Section 3 we explain our attack and its theoretical background. Section 4 and Section 5 provide results from simulations and conclusions from the power analysis of an exemplary implementation on a standard microcontroller. The paper ends with possible countermeasures and final conclusions.

2 Prime Generation

In this section we have a closer look at the prime generation step. Moreover, we formulate assumptions on the side-channel leakage that are relevant for the next sections.

Definition 1. *For any $k \in N$ a* k-bit integer *denotes an integer that is contained in the interval* $[2^{k-1}, 2^k)$*. For a positive integer $m \geq 2$ as usually $Z_m := \{0, 1, \ldots, m-1\}$ and $Z_m^* := \{x \in Z_m \mid \gcd(x, m) = 1\}$. Further, $b (\bmod m)$ denotes that element in Z_m that has the same m-remainder as b.*

Pseudoalgorithm 1 (prime generation)

```
1) Generate a (pseudo-)random odd integer $v \in [2^{k-1}, 2^k)$
2) Check whether $v$ is prime. If $v$ is composite then goto Step 1
3) $p := v$ (resp., $q := v$)
```

Pseudoalgorithm 1 represents the most straight-forward approach to generate a random k-bit prime. In Step 2 trial divisions by small odd primes from a particular set $\mathcal{T} := \{r_2, \ldots, r_N\}$ are performed, and to the 'surviving' prime candidates the Miller-Rabin primality test (or, alternatively, any other probabilistic primality test) is applied several times. The 'trial base' $\mathcal{T} := \{r_2, \ldots, r_N\}$ (containing all odd primes $\leq$ some bound B) should be selected to minimize the average run-time of Step 2.

By the prime number theorem

$$\#\text{ primes } \in [2^{k-1}, 2^k) \approx \frac{2^k}{\log_e(2^k)} - \frac{2^{k-1}}{\log_e(2^{k-1})} = \frac{2^{k-1}}{\log_e(2)} \left(\frac{2}{k} - \frac{1}{k-1} \right). \quad (1)$$

Consequently, for a randomly selected odd integer $v \in [2^{k-1}, 2^k)$ we obtain

$$\text{Prob}\left(v \in [2^{k-1}, 2^k) \text{ is prime}\right) \approx \frac{2}{\log_e(2)} \left(\frac{2}{k} - \frac{1}{k-1} \right) \approx \frac{2}{k \log_e(2)}. \quad (2)$$

For $k = 512$ and $k = 1024$ this probability is $\approx 1/177$ and $\approx 1/355$, respectively. This means that in average 177, resp. 355, prime candidates have to be checked to obtain a k-bit prime. The optimal size $|\mathcal{T}|$ depends on k and on the ratio between the run-times of the trial divisions and the Miller-Rabin tests. This ratio clearly is device-dependent.

Hence Pseudoalgorithm 1 requires hundreds of calls of the RNG (random number generator), which may be too time-consuming for many applications. Pseudoalgorithm 2 below overcomes this problem as it only requires one k-bit random number per generated prime. References [2, 11], for example, thus recommend the successive incrementation of the prime candidates or at least mention this as a reasonable option. Note that the relevant part of Pseudoalgorithm 2 matches with Algorithm 2 in [2] (cf. also the second paragraph on p. 444). The parameter t in Step 2d depends on the tolerated error probability.

Pseudoalgorithm 2 (prime generation)

```
1)  Generate a (pseudo-)random odd integer $v_0 \in [2^{k-1}, 2^k)$
          $v := v_0$;
2)  a)    $i := 2$;
    b)    while ($i \le N$) do {
            if ($r_i$ divides $v$) then {
              $v := v + 2$; GOTO Step 2a; }
            $i^{++}$;
          }
    c)    $m := 1$;
    d)    while ($m \le t$) do {
            apply the Miller-Rabin primality test to $v$;
            if the primality test fails then {
              $v := v + 2$; GOTO Step 2a; }
            else $m^{++}$;
          }
3)        $p := v$ (resp., $q := v$)
```

Pseudoalgorithm 2 obviously ‘prefers’ primes that follow long prime gaps but until now no algebraic attack is known that exploits this property. However, the situation may change if side-channel analysis is taken into consideration. We formulate two assumptions that will be relevant for the following.

Assumptions 1. a) Pseudoalgorithm 2 is implemented on the target device.
b) Power analysis allows a potential attacker to identify for each prime candidate v after which trial division the while-loop in Step 2b terminates. Moreover, he is able to realize whether Miller-Rabin primality test(s) have been performed.

Remark 1. We may assume that
(i) a strong RNG is applied to generate the odd number v_0 in Step 1 of Pseudoalgorithm 2.
(ii) the trial division algorithm itself and the Miller-Rabin test procedure are effectively protected against side-channel attack. This means that the side-channel leakage does not reveal any information on the dividend of the trial divisions, i.e. on the prime candidates v.

Remark 2. (i) If any of the security assumptions from Remark 1 are violated it may be possible to improve our attack or to mount a different, even more efficient side-channel attack. This is yet outside the scope of this paper. In the following we merely exploit Assumption b)
(ii) Assumption b) is clearly fulfilled if the attacker is able to determine the beginning or the end of each trial divisions. If all trial divisions require essentially the same run-time (maybe depending on the prime candidates v) it suffices to identify the beginning of the while-loop or the incrementation by 2 in Step 2b. The run-time also reveals whether Miller-Rabin tests have been performed.
(iii) It may be feasible to apply our attack also against software implementations on PCs although power analysis is not applicable there. Instead, the attacker may try to mount microarchitectural attacks (cache attacks etc.).
(iv) We point out that more efficient (and more sophisticated) prime generation algorithms than Pseudoalgorithm 2 exist (cf. Section 6 and [1, 7, 11], Note 4.51(ii), for instance).

3 The Attack

In Section 3 we describe and analyze the theoretical background of our attack. Empirical and experimental results are presented in Section 4 and Section 5.

3.1 Basic Attack

We assume that the candidate $v_m := v_0 + 2m$ in Pseudoalgorithm 2 is prime, i.e. $p = v_m$. If for $v_j = v_0 + 2j$ Pseudoalgorithm 2 returned to Step 2a after the trial division by r_i then v_j is divisible by r_i. This gives

$$\left.\begin{array}{rl} v_j \equiv & 0 \pmod{r_i} \\ v_j = & v_0 + 2j \\ p = & v_m = v_0 + 2m \end{array}\right\} \Rightarrow p = v_j + 2(m-j) \equiv 2(m-j) \pmod{r_i}. \qquad (3)$$

Let

$$\mathcal{S}_p := \{2\} \cup \{r \in \mathcal{T} \mid \text{division by } r \text{ caused a return to 2a for at least one } v_j\}. \qquad (4)$$

We point out that 'caused a return ...' is not equivalent to 'divides at least one v_j'. (Note that it may happen that $r \in \mathcal{T} \setminus \mathcal{S}_p$ divides some v_j but loop 2b terminates earlier due to a smaller divisor r' of v_j.) We combine all equations of type (3) via the Chinese Remainder Theorem (CRT). This yields a congruence

$$a_p \equiv p (\operatorname{mod} s_p) \qquad \text{for} \qquad s_p := \prod_{r \in \mathcal{S}_p} r \tag{5}$$

with known a_p. As $pq = n$ we have

$$a_q :\equiv q \equiv a_p^{-1} n (\operatorname{mod} s_p). \tag{6}$$

By observing the generation of q we obtain

$$b_q \equiv q (\operatorname{mod} s_q) \qquad \text{and} \qquad b_p \equiv p \equiv b_q^{-1} n (\operatorname{mod} s_q) \tag{7}$$

where $\mathcal{S}_q$ and s_q are defined analogously to $\mathcal{S}_p$ and s_p. Equations (5), (6) and (7) give via the CRT integers c_p, c_q and s with

$$s := \operatorname{lcm}(s_p, s_q), \;\; c_p \equiv p (\operatorname{mod} s), \;\; c_q \equiv q (\operatorname{mod} s) \quad \text{and} \quad 0 \le c_p, c_q < s. \tag{8}$$

By (8)

$$p = s x_p + c_p \quad \text{and} \quad q = s y_q + c_q \quad \text{with unknown integers} \quad x_p, y_q \in \mathbb{N} \tag{9}$$

while c_p, c_q and s are known. Lemma 1 transforms the problem of finding p and q into a zero set problem for a bivariate polynomial over Z.

Lemma 1. *(i) The pair (x_p, y_q) is a zero of the polynomial*

$$f \colon Z \times Z \to Z, \;\; f(x,y) := sxy + c_p y + c_q x - t \;\; \textit{with} \;\; t := (n - c_p c_q)/s. \tag{10}$$

(ii) In particular

$$t \in \mathbb{N}, \qquad \text{f } \textit{is irreducible over } \mathrm{Z}, \qquad \textit{and} \tag{11}$$

$$0 < x_p, y_q < \max\left\{\frac{p}{s}, \frac{q}{s}\right\} < \frac{2^k}{s}. \tag{12}$$

Proof. Obviously,

$$0 = pq - n = (sx_p + c_p)(sy_q + c_q) - n = s^2 x_p y_q + s c_p y_q + s c_q x_p - (n - c_p c_q),$$

which verifies (i). Since $n \equiv c_p c_q (\operatorname{mod} s)$ the last bracket is a multiple of s, and hence $t \in \mathrm{Z}$. Since $c_p \equiv p \not\equiv 0 (\operatorname{mod} r_j)$ and $c_q \equiv q \not\equiv 0 (\operatorname{mod} r_j)$ for all prime divisors r_j of s we conclude $\gcd(s, c_p) = \gcd(s, c_q) = 1$, and in particular $\gcd(s, c_p, c_q, t) = 1$. Assume that $f(x,y) = (ax + by + c)(dx + ey + f)$ for suitably selected integers a, b, c, d, e and f. Comparing coefficients immediately restricts to $(a = e = 0)$ or $(b = d = 0)$. The gcd-properties yield $\gcd(bd, bf) = 1 = \gcd(bd, cd)$, resp. $\gcd(ae, af) = 1 = \gcd(ae, ce)$, and thus $b = d = 1$, resp. $a = e = 1$, leading to a contradiction. Assertion (12) is obvious.

In general finding zeroes of bivariate polynomials over Z is difficult. It is well-known that 'small' integer solutions yet can be found efficiently with the LLL-algorithm, which transforms the zero set problem to finding short vectors in lattices.

Theorem 1. *(i) Let $p(x,y)$ be an irreducible polynomial in two variables over* Z*, of maximum degree δ in each variable separately. Let X, Y be upper bounds for the absolute value of the searched solutions x_0, y_0. Define $\tilde{p}(x,y) := p(xX, yY)$ and let W be the absolute value of the largest coefficient of $\tilde{p}$. If*

$$XY < W^{2/(3\delta)}$$

then in time polynomial in $(logW, \delta)$, one can find all integer pairs (x_0, y_0) with $p(x_0, y_0) = 0$, $|x_0| < X$,$|y_0| < Y$.
(ii) Let p and q be k-bit primes and $n = p \cdot q$. If integers s and c_p are given with $s \geq 2^{\frac{k}{2}}$ and $c_p \equiv p(\bmod s)$ then one can factorize n in time polnomial in k.

Proof. (i) [5], Corollary 2
(ii) We apply assertion (i) to the polynomial $f(x,y)$ from Lemma 1. By (12) we have $0 < x_p < X := 2^k/s$ and $0 < y_q < Y := 2^k/s$. Let $\tilde{f}(x,y) = f(xX, yY)$ and let W denote the maximum of the absolute values of the coefficients of $\tilde{f}(x,y)$. Then $W \geq sXY = \frac{2^{2k}}{s}$, and for $s > 2^{\frac{k}{2}}$ we get

$$XY = \left(\frac{2^k}{s}\right)^2 < \left(\frac{2^{2k}}{s}\right)^{\frac{2}{3}} \leq W^{\frac{2}{3}}$$

where the first inequality follows from $2^k < s^2$ by some equivalence transformations. Since the degree δ in each variable is one by (i) we can find (x_p, y_q) in time polynomial in k.

3.2 Gaining Additional Information

Theorem 1 demands $\log_2(s) > 0.5k$. If $\log_2(s)$ is only slightly larger than $0.5k$ the dimension of the lattice ($\rightarrow$ LLL-algorithm) has to be very large which affords much computation time. For concrete computations thus $\log_2(s) \geq C > 0.5k$ is desirable for some bound C that is reasonably larger than $0.5k$.

If $\log_2(s) \geq C$ Theorem 1 can be applied and then the work is done. If $\log_2(s) < C$ one may multiply s by some relatively prime integer s_1 (e.g. the product of some primes in $\mathcal{T} \setminus (\mathcal{S}_p \cup \mathcal{S}q)$) with $\log_2(s) + \log_2(s_1) > C$. Of course, the adversary has to apply Theorem 1 to any admissible pair of remainders $(p(\bmod{(s \cdot s_1)}), q(\bmod{(s \cdot s_1)}))$. Theorem 1 clearly yields the factorization of n only for the correct pair $(p(\bmod{(s \cdot s_1)}), q(\bmod{(s \cdot s_1)}))$, which increases the workload by factor 2^{s_1}. Note that $p(\bmod{(s \cdot s_1)})$ determines $q(\bmod{(s \cdot s_1)})$ since $n(\bmod{(s \cdot s_1)})$ is known.

The basic attack explained in Subsection 3.1 yet does not exploit all information. Assume that the prime $r_u \in \mathcal{T}$ does not divide s, which means that

$r_u \notin \mathcal{S}_p \cup \mathcal{S}_q$ or, equivalently, that the trial division loop 2b in Algorithm 2 has never terminated directly after a division by r_u. Assume that during the search for p the prime candidates $v_{j_1}, \ldots, v_{j_\tau}$ have been divided by r_u. Then

$$\left.\begin{aligned} v_{j_1} &= p - 2(m - j_1) \not\equiv 0 \pmod{r_u} \\ &\vdots \\ v_{j_\tau} &= p - 2(m - j_\tau) \not\equiv 0 \pmod{r_u} \\ v_m &= p \qquad\qquad\quad \not\equiv 0 \pmod{r_u} \end{aligned}\right\} \Longrightarrow \tag{13}$$

$$p \not\equiv 0, 2(m-j_1), \ldots, 2(m-j_\tau) \pmod{r_u}. \tag{14}$$

This yields a 'positive list'

$$L'_p(r_u) = \{0, 1, \ldots, r_u - 1\} \setminus \{0, 2(m-j_1)(\bmod r_u), \ldots, 2(m-j_\tau)(\bmod r_u)\} \tag{15}$$

of possible r_u-remainders of p. Analogously, one obtains a positive list $L'_q(r_u)$ for possible r_u-remainders of q. The relation $p \equiv nq^{-1}(\bmod r_u)$ reduces the set of possible r_u-remainders of p further to

$$L_p(r_u) := L'_p(r_u) \cap \left(nL'_q(r_u)^{-1}(\bmod r_u)\right), \quad \text{and finally} \tag{16}$$

$$(p(\bmod r_u), q(\bmod r_u)) \in \{(a, na^{-1}(\bmod r_u)) \mid a \in L_p(r_u)\}. \tag{17}$$

For prime r_u equations (16) and (17) provide

$$I(r_u) := \log_2\left(\frac{r_u}{|L_p(r_u)|}\right) = \log_2(r_u) - \log_2(|L_p(r_u)|) \tag{18}$$

bit of information. From the attacker's point of view the most favourable case clearly is $|L_p(r_u)| = 1$, i.e. $I(r_u) = \log_2(r_u)$, which means that $p(\bmod r_u)$ is known. The attacker may select some primes $r_{u_1}, \ldots, r_{u_w} \in \mathcal{T} \setminus (\mathcal{S}_p \cup \mathcal{S}_q)$ that provide much information $I(r_{u_1}), \ldots, I(r_{u_w})$ (or, maybe more effectively, selecting primes with large ratios $I(r_{u_1})/\log_2(r_{u_1}), \ldots, I(r_{j_w})/\log_2(r_{j_w})$) where w clearly depends on the gap $C - \log_2(s)$. Then he applies Theorem 1 to $(s \cdot s_1)$ with $s_1 = r_{u_1} \cdots r_{u_w}$ for all $|L_p(r_{u_1})| \cdots |L_p(r_{u_w})|$ admissible pairs of remainders $(p(\bmod (s \cdot s_1)), q(\bmod (s \cdot s_1)))$. Compared to a 'blind' exhaustive search without any information $p(\bmod s_1)$ (17) reduces the workload by factor $2^{I(r_{u_1})+\cdots+I(r_{u_w})}$ or, loosely speaking, reduces the search space by $I(r_{u_1}) + \cdots + I(r_{u_w})$ bit.

After the primes p and q have been generated the secret exponents $d_p \equiv e^{-1}(\bmod (p-1))$ and $d_q \equiv e^{-1}(\bmod (q-1))$ are computed. These computations may provide further information. Analogously to Assumptions 1 we formulate

Assumptions 2. The exponents d_p and d_q are computed with the extended Euclidean algorithm, and the power consumption allows the attacker to determine the number of steps that are needed by the Euclidean algorithm.

We may clearly assume $p > e$, and thus the first step in the Euclidean algorithm reads $p - 1 = \alpha_2 e + x_3$ with $\alpha_2 \geq 1$ and $x_3 = p - 1(\bmod e)$. The following steps depend only on the remainder $p-1(\bmod e)$, resp. on $p(\bmod e)$. As $p \not\equiv 0(\bmod e)$ for $j \in \mathbb{N}_0$ we define the sets

$$M'(j)=\{x \in \mathrm{Z}_{\mathrm{e}}^* \mid \text{for (e, x−1) the Euclidean alg. terminates after j steps}\}. \quad (19)$$

Assume that the attacker has observed that the computation of d_p and d_q require v_p and v_q steps, respectively. By definition, $p(\bmod e) \in M'(v_p-1)$ and $q(\bmod e) \in M'(v_q-1)$. Similarly as above

$$M_p := M'(v_p-1) \cap \Big(n\,(M'(v_q-1))^{-1}\,(\bmod e)\Big), \quad \text{and finally} \quad (20)$$

$$(p(\bmod e), q(\bmod e)) \in \{(a, na^{-1}(\bmod e)) \mid a \in M_p\}. \quad (21)$$

If e is relatively prime to s, resp. to $s \cdot s_1$ (e.g. for $e = 2^{16}+1$), the attacker may apply Theorem 1 to $s \cdot e$, resp. to $s \cdot s_1 \cdot e$, and the gain of information is $I(e) = \log_2(e) - \log_2(|M_p|)$. If $\gcd(s,e) > 1$, resp. if $\gcd(s \cdot s_1, e) > 1$, one uses $e' := e/\gcd(s,e)$, resp. $e' := e/\gcd(s \cdot s_1, e)$, in place of e.

Remark 3. In Section 2 we assumed $p, q \in [2^{k-1}, 2^k)$. We note that our attack merely exploits $p, q < 2^k$, and it also applies to unbalanced primes p and q.

4 Empirical Results

The basic basic attack reveals congruences $p(\mod s)$ and $q(\mod s)$ for some modulus s. If s is sufficiently large Theorem 1 allows a successful attack. The term $\log_2(s)$ quantifies the information we get from side-channel analysis. The product s can at most equal $\prod_{r \in \mathcal{T}} r$ but usually it is much smaller. Experiments show that the bitsize of s may vary considerably for different k-bit starting candidates v_0 for Pseudoalgorithm 2. Theorem 1 demands $\log_2(s) > \frac{k}{2}$, or (from a practical point of view) even better $\log_2(s) \geq C$ for some bound C which allows to apply the LLL-algorithm with moderate lattice dimension. We investigated the distribution of the bitlength of s for $k = 512$ and $k = 1024$ bit primes and for different sizes of the trial division base $\mathcal{T} = \{r_2 = 3, 5, 7, \ldots, r_N\}$ by a large number of simulations. Note that two runs of Pseudoalgorithm 2 generate an RSA modulus n of bitsize $2k-1$ or $2k$.

We implemented Pseudoalgorithm 2 (with $t = 20$ Miller-Rabin-tests) in MAGMA [10] and ran the RSA-generation process 10000 times for each of several pairs (k, N). For $k = 512$ and $N = 54$ we obtained the empirical cumulative distribution shown in Figure 1. The choice $N = 54$ is natural since $r_{54} = 251$ is the largest prime smaller than 256, and thus each prime of the trial division base can be represented by one byte. Further results for $k = 512$ and $k = 1024$ are given in Table 1, resp. in Table 2. The ideas from Subsection 3.2 are considered below.

Further on, we analysed how the run-time of the LLL-algorithm and thus the run-time of the factorization of the RSA modulus n by Theorem 1 depends

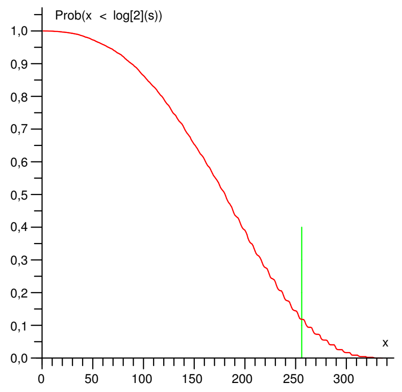

Fig. 1. Basic attack: Cumulative distribution for $r_N = 251$ and $k = 512$ (1024-bit RSA moduli)

$k = 512$				
N	r_N	$Prob(\log_2(s) > 256)$	$Prob(\log_2(s) > 277)$	$\log_2(\prod_{r \le r_N} r)$
54	251	0.118	0.055	334.8
60	281	0.188	0.120	388.3
70	349	0.283	0.208	466.5

Table 1. Basic attack

on the bitsize of s. We did not use the original algorithm of Coppersmith from [5]. Instead we implemented Coron's algorithm from [6] in the computer algebra system MAGMA [10].

The LLL-reduction in Coron's algorithm uses a lattice of dimension $\omega = (\tilde{k} + \delta)^2 - \tilde{k}^2$ where δ denotes the degree of the polynomial and $\tilde{k}$ an adjustable parameter of the algorithm. In our case $\delta = 1$, so the lattice dimension is $\omega = 2\tilde{k} + 1$. For $\omega = 15$ our implementation (i.e. the LLL-substep) never terminated in less than one hour; we stopped the process in these cases. Table 3 provides empirical results. (More sophisticated implementations may allow to get by with smaller s ($\rightarrow$ larger ω) but this is irrelevant for the scope of this paper.)

If the basic attack yields $\log_2(s) \le k/2$ or $\log_2(s) < C$ the attacker may apply the techniques from Subsection 3.2. Since the information $I(e)$ is deduced from the computation of $\gcd(e, p-1)$ it is independent of the outcome of the basic attack while $I(r_{u_1}) + \cdots + I(r_{u_w})$ depends on the size of s. If $\log_2(s)$ is contained in $[230, 260]$, a relevant range for $k = 512$, for $r_N = 251$ the mean value of $I(r_{u_1}) + \cdots + I(r_{u_w})$ is nearly constant.

Simulations for $k = 512$, $e = 2^{16} + 1$, and $\mathcal{T} = \{3, \ldots, 251\}$ show that $(I(2^{16} + 1), \log_2(2^{16} + 1)) = (6.40, 16)$, while $(I(r_{u_1}), \log_2(r_{u_1})) = (2.81, 6.39)$,

$k = 1024$				
N	r_N	$Prob(\log_2(s) > 512)$	$Prob(\log_2(s) > 553$	$\log_2(\prod_{r \le r_N} r)$
100	541	0.125	0.065	729.7
110	601	0.178	0.113	821.2
120	659	0.217	0.150	914.5

Table 2. Basic attack

$Bitsize(n = pq)$	ω	$Min\{Bitsize(s) \vert {}^{factorization}_{succesfull}\}$	$\approx run - time(sec)$
512	5	156	0.01
512	7	148	0.07
512	9	144	0.24
512	11	141	1.1
512	13	139	4.6
1024	5	308	0.02
1024	7	294	0.17
1024	9	287	0.66
1024	11	281	3.1
1024	13	277	13.2
1536	5	462	0.05
1536	7	440	0.36
1536	9	428	1.6
1536	11	420	8.1
1536	13	415	41.5
2048	5	616	0.06
2048	7	587	0.76
2048	9	571	3.3
2048	11	560	18.5
2048	13	553	87.4

Table 3. Empirical run-times for different lattice dimensions and moduli

resp. $(I(r_{u_1}) + I(r_{u_2}), \log_2(r_{u_1}) + \log_2(r_{u_2})) = (4.80, 13.11)$, resp. $(I(r_{u_1}) + I(r_{u_2}) + I(r_{u_3}), \log_2(r_{u_1}) + \log_2(r_{u_2}) + \log_2(r_{u_3})) = (6.42, 20.04)$, where r_{u_1}, r_{u_2} and r_{u_3} (in this order) denote those primes in $\mathcal{T} \setminus (\mathcal{S}_p \cup \mathcal{S}_q)$ that provide maximum information.

Multiplying the modulus s from the basic attack by e, resp. by $e \cdot r_{u_1}$, resp. by $e \cdot r_{u_1} \cdot r_{u_2}$, resp. by $e \cdot r_{u_1} \cdot r_{u_2} \cdot r_{u_3}$, increases the bitlength of the modulus by 16 bits, resp. by $16 + 6.39 = 22.39$, resp. by 29.11, resp. by 36.04 in average although the average workload increases only by factor $2^{16-6.40} = 2^{9.6}$, resp. by $2^{9.6} \cdot 2^{6.39-2.81} = 2^{13.18}$, resp. by $2^{17.91}$, resp. by $2^{23.22}$.

Combining this with the run-time of 13.2 seconds given in Table 3 with our implementation we can factorize a 1024-Bit RSA modulus in at most $2^{13.18} \cdot 13.2\ sec \approx 34\ hours$ (in about 17 hours in average) if the modulus s gained by the basic attack only consists of $277 - 22 = 255$ bits. According to our experiments this happens with probability ≈ 0.119. Table 1 shows that the methods of Subsection 3.2 double (for our LLL-implementation) the success probability. Further on we want to emphasize that by Theorem 1 the attack becomes principally feasible if the basic attack yields $\log_2(s) > 256$. So, at cost of increasing the run-time by factor $2^{13.18}$ the modulus n can be factored with the LLL-algorithm if the basic attack yields a modulus s with $\log_2(s) > 256 - 22 = 234$. This means that the success probability would increase from 11.8% to 21.2% (cf. Table 1).

5 Experimental Results

The central assumption of our attack is that the power consumption reveals the exact number of trial divisions for the prime candidates $v_0 = v, v_1 = v + 2, \ldots$. To verify that this assumption is realistic we implemented the relevant part of Pseudoalgorithm 2 (Step 1 to Step 2b) on a standard microcontroller (Atmel ATmega) and conducted measurements.

The power consumption was simply measured as a voltage drop over a resistor that was inserted into the GND line of this chip. An active probe was used. As the controller is clocked by its internal oscillator running at only 1MHz a sampling-rate of 25 MHz was sufficient. The acquired waveforms were high-pass-filtered and reduced to one peak value per clock cycle.

Figure 2 shows the empirical distribution of the number of clock cycles per trial division. We considered 2000 trial divisions $testdiv(v, r)$ with randomly selected 512 bit numbers v and primes $r < 2^{16}$. The number of clock cycles are contained in the interval $[24600, 24900]$, which means that they differ not more than about 0.006μ cycles from their arithmetic mean μ. In our standard case $\mathcal{T} = \{3, \ldots, 251\}$ a maximum sequence of 53 consecutive trial divisions may occur. We point out that it is hence not necessary to identify the particular trial divisions, it suffices to identify those positions of the power trace that correspond to the incrementation of the prime candidates by 2. Since short and long run-times of the individual trial divisions should compensate to some extent, this conclusion should remain valid also for larger trial bases and for other implementations of the trial divisions with (somewhat) larger variance of the run-times.

The crucial task is to find characteristic parts of the power trace that allow to identify the incrementation operations or even the individual trial divisions. The trial division algorithm and the incrementation routine were implemented in a straight-forward manner in an 8-bit arithmetic. Since the incrementation operations leave the most significant parts of the prime candidates $v_0, v_1, \ldots$ unchanged and since all divisors are smaller than 255 it is reasonable to expect that the power consumption curve reveals similar parts. Observing the following sequence of operations confirmed this conjecture.

Prime generation and trial divisions

```
rnd2r ();    // generates an odd 512 bit random number v
testdiv512 (v,3);   // trial division by 3
```

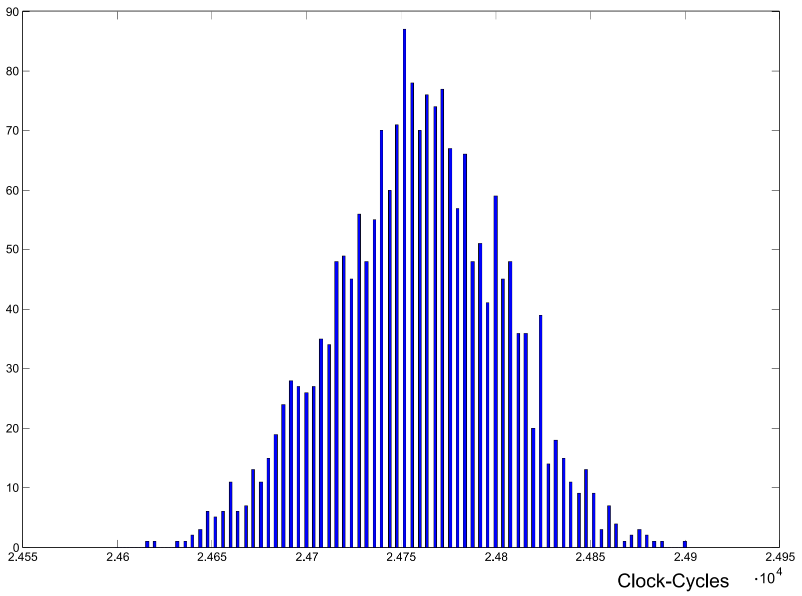

Fig. 2. Empirical run-times of trial divisions

```
testdiv512 (v,5);
testdiv512 (v,7);
incrnd (v);          // increments v by 2
testdiv512 (v,3);
incrnd (v);
testdiv512 (v,3);
testdiv512 (v,5);
```

We measured the power-consumption x_i for each clock cycle i. We selected short sample sequences $\{y_1 = x_t, \ldots, y_M = x_{t+M-1}\} \subset \{x_1, x_2, \ldots\}$ that correspond to 10 to 20 consecutive cycles, and searched for similar patterns in the power consumption curve. For fixed sample pattern $(y_1, \ldots, y_M)$ we used the 'similarity function',

$$a_j = \frac{1}{M} \sum_{i=1}^{M} |x_{i+j} - y_i| \quad \text{for shift parameter } j = 1, \ldots, N - M, \tag{22}$$

which compares the sample sequence $(y_1, \ldots, y_M)$ with a subsequence of power values of the same length that is shifted by j positions. A small value a_j indicates that $(x_j, \ldots, x_{j+M-1})$ is 'similar' to the sample sequence $(y_1, \ldots, y_M)$. It turned out that it is even more favourable to consider the minimum within 'neighbourhoods' rather than local minima. More precisely, we applied the values

$$b_j = \min\{a_j, \ldots, a_{j+F-1}\} \tag{23}$$

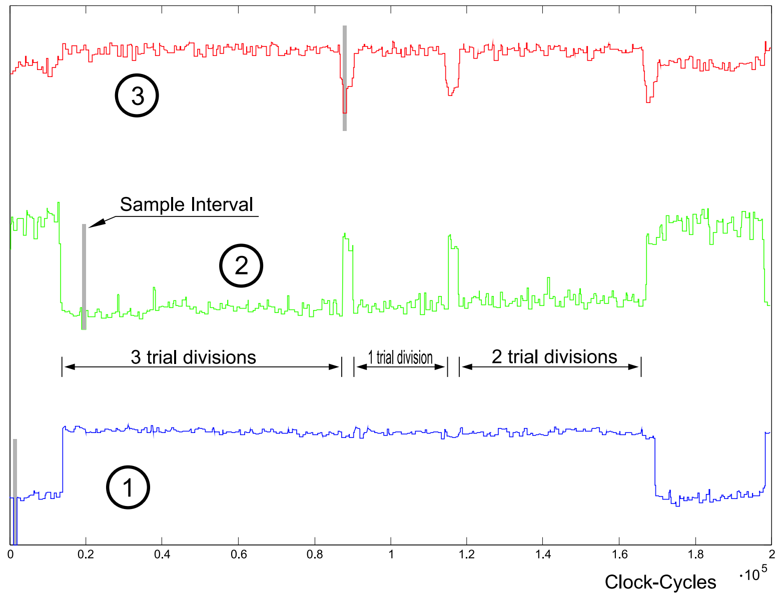

Fig. 3. Similarity curves (b_j-values)

with $F \approx 100$. Figure 3 shows three graphs of b_j-values. The vertical grey bars mark the position of the selected sample sequence $(y_1, \ldots, y_M)$. For Curve (1) the sample sequence was part of the random number generation process. Obviously, this sample sequence does not help to identify any trial division or the incrementation steps. For Curve (2) we selected a sample sequence within a trial division. The high peaks of Curve (2) stand for 'large dissimilarity' and identify the incrementation steps. Curve (3) shows the b_j-values for a sample sequence from the incrementation step, and low peaks show the positions of the incrementation steps. Our experiments showed that the procedure is tolerant against moderate deviations of M, F and the sample pattern $(y_1, \ldots, y_M)$ from the optimal values.

6 Countermeasures and Alternative Implementations

Our attack can be prevented by various countermeasures. The most rigorous variants are surely to divide each prime candidate by all elements of the trial base or to generate each prime candidate independent from its predecessors ($\rightarrow$ Pseudoalgorithm 1). However, both solutions are very time-consuming and thus may be inappropriate for many applications due to performance requirements. Clearly more efficient is to XOR some fresh random bits to every τ^{th} prime candidate v in order to compensate the side-channel leakage of the trial divisions of the previous τ prime candidates. These random bits should at least compensate the average information gained from the leakage or, even better, compensate the maximum information leakage that is possible (worst-case scenario). In analogy to (5) let s_τ denote the product of all primes of $\mathcal{T}$, after which the while loop in Step 2b of Pseudoalgorithm 2 has terminated for at least one of the last τ prime candidates v_j. For $\tau = 10$, for instance, the while-loop must have terminated at least three times after the trial division by 3 and at least once after a trial division by 5. In the worst case, the remaining six loops terminated after the division by one of the six largest primes of $\mathcal{T}$, which gives the (pessimistic) inequality $\log_2(s_\tau) \leq \log_2(3 \cdot 5 \cdot r_{N-5} \cdot \cdots \cdot r_N)$. For $k = 512$ and $\mathcal{T} = \{3, 5, \ldots, 251\}$, for instance, $\log_2(s_{10}) < \log_2(3 \cdot 5 \cdot 227 \cdot \ldots \cdot 251) \approx 51.2$. Simulations showed that the average value of $\log_2(s_{10})$ is much smaller (≈ 18.6). By applying the ideas from Subsection 3.2 the attacker may gain some additional information, in the worst case yet not more than $\sum_{j=4}^{48} \log_2(r_j/(r_j - 6)) \approx 10.66$ bit. The designer should be on the safe side if he selects randomly at least 8 bytes of each 10^{th} prime candidate v and XORs 64 random bits to these positions. (Simulations indicate that the average overall gain of information is less than 24 bit.)

We mention that several other, more sophisticated prime generation algorithms have been proposed in literature. For instance, the remainders of the first prime candidate v_0 with respect to all primes in $\mathcal{T}$ may be stored in a table, saving the trial divisions for all the following prime candidates in favour of modular additions of all table entries by 2 ([11], Note 4.51 (ii)). This prevents our attack but, of course, a careless implemention of the modular additions may also reveal side-channel information. A principal disadvantage is that a large table has to be built up, stored and managed, which may cause problems in devices with little resources. Alternatively, in a first step one may determine an integer v that is relatively prime to all primes of a trial base $\mathcal{T}$. The integer v then is increased by a multiple of the product of all primes from the trial base until v is a prime. Reference [3] applies techniques from [1], which were originally designated for the shared generation of RSA keys. The authors of [3] yet point out that they do not aim at good performance, and in fact, for many applications performance aspects are crucial.

Reference [7] proposes a prime generation algorithm that uses four integer parameters P (large odd number, e.g. the product of the first $N-1$ odd primes), w, and $b_{min} \leq b_{max}$. The algorithm starts with a randomly selected integer $y_0 \in \mathrm{Z}_\mathrm{P}^*$ and generates prime candidates $v_j = v_j(y_j) = (v+b)P + y_j$ or $v_j(y_j) = (v+b+1)P - y_j$, respectively, for some integer $b \in [b_{min}, b_{max}]$ and $y_j = 2^j y_0 (\bmod P)$ until a prime is found, or more precisely, until some v_j passes the primality tests. Reference [4] describes a (theoretical) side channel attack on a special case of this scheme (with $b_{min} = b_{max} = 0$ and known (P, w)) on a careless implementation that reveals the parity of the prime candidates v_j. This attack is successful in about 0.1% of the trials, and [4] also suggests countermeasures. For further information on primality testing we refer the interested reader to the relevant literature.

7 Conclusion

This paper presents an elementary side-channel attack which focuses on the RSA key generation. The attack works under weak assumptions on the side-channel leakage, and practical experiments show that these assumption may be realistic. If the attack is known it can be prevented effectively.

Reference [4] and the above results demonstatrate that the RSA key generation process may be vulnerable to side-channel attacks. It appears to be reasonable to analyse implementations of various key generation algorithms in this regard. New attacks (possibly in combination with weaker assumptions than Remark 1) and in particular effective countermeasures may be detected.

References

1. D. Boneh. J. Franklin: Efficient Generation of Shared RSA keys. In: B.S. Kaliski (ed.): Advances in Cryptology — Crypto 97, Springer, Lecture Notes in Computer Science 1294, Berlin 1997, 425–439.
2. Brandt, I. Damgard, P. Landrock: Speeding up Prime Number Generation. In: H. Imai, R. Rivest, T. Matsumoto (Hrsg.): Asiacrypt 91. Springer, Lecture Notes in Computer Science 739, Springer, Berlin 1993, 440–449.
3. H. Chabanne, E. Dottax, L. Ramsamy: Masked Prime Number Generation. Generation.https://www.cosic.esat.kuleuven.be/wissec2006/papers/29.pdf
4. C. Clavier, J.-S. Coron: On the Implementation of a Fast Prime Generation Algorithm. In: P. Paillier, I. Verbauwhede (eds.): CHES 2008, Springer, Lecture Notes In Computer Science 4727, Berlin 2007, 443–449.
5. D. Coppersmith: Small Solutions to Polynomial Equations, and Low Exponent Vulnerabilities. J. Crypt. 10(4), 1997, 233-260.
6. J.S. Coron: Finding Small Roots of Bivariate Integer Polynomial Equations: A Direct Approach. In: A. Menezes (ed.): Advances in Cryptology – CRYPTO 2007, Springer, Lecture Notes In Computer Science 4622, Berlin 2007, 379–394.
7. M. Joye, P. Paillier: Fast Generation of Prime Numbers on Portable Devices. An Update. In: L. Goubin, M. Matsui (eds.): CHES 2006, Springer, Lecture Notes in Computer Science 4249, Berlin 2006, 160–173.
8. P. Kocher: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS and Other Systems. In: N. Koblitz (ed.): Crypto 1996, Springer, Lecture Notes in Computer Science 1109, Heidelberg 1996, 104–113.
9. P. Kocher, J. Jaffe, B. Jun: Differential Power Analysis. In: M. Wiener (ed.): Advances in Cryptology — CRYPTO '99, Springer, LNCS 1666, Berlin 1999, 388–397.
10. Magma, Computational Algebra Group, School of Mathematics and Statistics, University of Sydney
11. A. Menezes, P. C. van Oorschot, S. A. Vanstone, Handbook of Applied Cryptography, CRC Press, Boca Raton, 1997.
12. W. Schindler: A Timing Attack against RSA with the Chinese Remainder Theorem. In: Ç.K. Koç, C. Paar (eds.): Cryptographic Hardware and Embedded Systems — CHES 2000, Springer, Lecture Notes in Computer Science 1965, Berlin 2000, 110–125.
13. W. Schindler: A Combined Timing and Power Attack. In: P. Paillier, D. Naccache (eds.): Public Key Cryptography — PKC 2002, Springer, Lecture Notes in Computer Science 2274, Berlin 2002, 263–279.