part 2

# Randomness and Key Generation

# 7

# Prime Number Generation and RSA Keys

**Marc JOYE and Pascal PAILLIER**

*Zama, Paris, France*

## 7.1. Introduction

A positive integer $q$ is said to be *prime* if $q > 1$ and if $q$ has no positive divisors except 1 and $q$.

Numerous cryptographic primitives rely on prime numbers, a good representative being the RSA cryptosystem used for encryption or digital signatures. Let $\mathcal{M}$ denote the message space. In its simplest form, the RSA cryptosystem requires two distinct primes $p$ and $q$ to form a modulus $N = pq$, an exponent $e$ that is co-prime with $\lambda(N) = \mathrm{lcm}(p-1, q-1)$,[1] and an injective (randomized) padding function $\mu\colon \mathcal{M} \to \mathbb{Z}_N$. There is also an exponent $d$ satisfying $ed \equiv 1 \pmod{\lambda(N)}$. Modulus $N$ and exponent $e$ are made public while exponent $d$ is kept private. A message $m \in \mathcal{M}$ is encrypted as $C \xleftarrow{\$} \mu(m)^e \bmod N$. The secrecy of primes $p$ and $q$ is primordial to guarantee the security as they allow recovering $d \leftarrow e^{-1} \bmod \mathrm{lcm}(p-1, q-1)$. Indeed, from $d$, the plaintext message $m$ can be recovered in two steps as $m^* \leftarrow C^d \bmod N$ and $m \leftarrow \mu^{-1}(m^*)$. For digital signatures, the roles of $e$ and $d$ are exchanged. For a deterministic padding function $\mu$, the signature $\sigma$ on a message $m \in \mathcal{M}$ is given by

1. LCM denotes the 'lowest common multiple'; in particular, $\mathrm{lcm}(p-1, q-1) = (p-1)(q-1)/\gcd(p-1, q-1)$.

$\sigma \leftarrow \mu(m)^d \bmod N$. The validity of $\sigma$ is then verified by checking that $\sigma^e \equiv \mu(m) \pmod N$ using public exponent $e$.

A naïve way to produce an $n$-bit prime consists in generating an odd $n$-bit integer and testing it for primality. The process is iterated until a prime is found. The expected number of trials is asymptotically equal to $(\ln 2^n)/2 \approx 0.347n$. Doing so, generating a random $1024$-bit prime thus requires about $355$ trials on average. The naïve prime generator can be made more efficient by selecting $n$-bit integers that are already co-prime with small primes, instead of just being co-prime with $2$. For example, one can define $\Pi$ as the product of the first $10$ primes, $\Pi = 2 \cdot 3 \cdot \cdots \cdot 29$, and randomly select an $n$-bit prime candidate $q$ satisfying $\gcd(q, \Pi) = 1$. The expected number of trials before a prime is found then drops heuristically to $(\ln 2^n)\frac{\varphi(\Pi)}{\Pi} \approx 0.109n$, where $\varphi$ denotes Euler's totient function. For $1024$-bit primes, this amounts to about $112$ trials. The complexity can be further reduced by including more primes in the definition of $\Pi$. This methodology however requires an efficient way to generate random $n$-bit integers co-prime with $\Pi$.

The rest of this chapter is devoted to describing efficient methods for producing random primes in a prescribed interval along those lines:

1) A prime candidate $q \in [q_{\text{min}}, q_{\text{max}}]$ is *constructively* generated so as to be co-prime with $\Pi$, a product of many (small) primes;

2) $q$ is tested for primality. If $q$ is not prime then it is updated in a way that its updated value remains co-prime with $\Pi$ and lies within $[q_{\text{min}}, q_{\text{max}}]$. This step is repeated until $q$ is found to be prime.

The output distribution of the primes that are generated also matters. In 2012, two independent teams of researchers collected RSA public keys from a wide variety of sources. Quite surprisingly, a non-negligible fraction of the collected RSA moduli exhibited a common prime factor. Sharing a common factor for non-duplicate RSA moduli completely compromises the security as calculating their greatest common divisor (GCD) reveals the secret factors and thus enables computing the private keys. These vulnerabilities apparently originated from the use of poor entropy in the generation of prime numbers $p$ and $q$ forming an RSA modulus $N = pq$. An important lesson is to generate RSA keys only after a proper initialization of the source of randomness. In particular, the initial random seed must be fresh and at least twice longer than the targeted security level.

The performance of algorithms highly depends on the hardware capabilities and specifics of the architecture implementing them. The methods developed in this chapter target embedded platforms allowing for super-fast evaluation of (modular) additions/subtractions/multiplications over large integers—which renders other types of

computations comparatively prohibitive in the absence of integrated hardware to support these. Examples include high-end smart cards equipped with an arithmetic crypto-coprocessor. In such a setting, algebraic tricks involving (modular) arithmetic operations on large numbers are largely preferred over glue instructions such as register switches, loop control, pointer management, etc.

REMARK. Cryptographic implementations should resist side-channel attacks as well as fault attacks. The different algorithms presented in this chapter are given in pseudo-code for the sake of clarity. Actual implementations should however ensure full security against these attacks, whose specifics are architecture-dependent. This *de facto* excludes schoolbook GCD algorithms, which leak a lot of information through the observation of their control flow. This chapter also assumes the availability of a secure cryptographic random number generator for producing uniformly random integers in a given range.

## 7.2. Primality Testing Methods

Primality testing has been an active research topic for many years. Computationally, two types of outputs are distinguished by nature: true primes and probable primes. The difference rests in the way these are generated. A *probable* prime (a.k.a. *pseudo-prime*) is usually obtained through a compositeness test, which is typically weaker but faster than a primality test. When such a test declares that a number is composite, then it is indeed with probability 1. However, if the test finds it to be prime, it is truly a prime with some probability $< 1$. Hence repeatedly running the test gives increasing confidence in the so-generated (probable) prime. Typical examples of compositeness tests include Fermat's test, the Solovay–Strassen test, and the Miller–Rabin test.

There also exist (actual) primality tests, which tell apart prime numbers from composite numbers with a strictly null error probability (e.g., Pocklington's test and its elliptic curve analogue, and the Jacobi sum test). These tests are generally more expensive and intricate to implement.

## 7.3. Generation of Random Units

Let $\mathbb{Z}_{\Pi}^*$ denote the set of integers modulo $\Pi$ that are co-prime with $\Pi$. The prime generation algorithms presented in this chapter require the random selection of an element $k \in \mathbb{Z}_{\Pi}^*$, that is, of a unit modulo $\Pi$. This section provides an algorithm that efficiently produces such an element with uniform output distribution. The design is based on the next two propositions, making use of Carmichael's function $\lambda$. Another

approach based on quadratic residuosity—simpler but not strictly uniform—is also described.

DEFINITION.– The *Carmichael function* $\lambda$ of an integer $\Pi \geq 2$, $\lambda(\Pi)$, is defined as the smallest positive integer $t$ such that $a^t \equiv 1 \pmod{\Pi}$ for every integer $a$ that is co-prime with $\Pi$.

In other terms, $\lambda(\Pi)$ denotes the exponent of the multiplicative group $\mathbb{Z}_{\Pi}^*$. Letting $\Pi = \prod_{i=1}^{L} {p_i}^{\delta_i}$ with $p_i$ prime and $\delta_i \geq 1$, it can be shown that

$$\lambda(\Pi) = \mathrm{lcm}(\lambda({p_1}^{\delta_1}), \ldots, \lambda({p_L}^{\delta_L}))$$

where

$$\lambda({p_i}^{\delta_i}) = \begin{cases} 2^{\delta_i - 2} & \text{if } p_i = 2 \text{ and } \delta_i > 2 \\ {p_i}^{\delta_i - 1}(p_i - 1) & \text{otherwise} \end{cases} .$$

PROPOSITION.– Let $\Pi > 1$ and let $k$ be an integer modulo $\Pi$. Then $k \in \mathbb{Z}_{\Pi}^*$ if and only if $k^{\lambda(\Pi)} \equiv 1 \pmod{\Pi}$.

*Proof.* This follows from the definition of Carmichael's function. If $k \in \mathbb{Z}_{\Pi}^*$ then $k^{\lambda(\Pi)} \equiv 1 \pmod{\Pi}$ since $\lambda(\Pi)$ is the exponent of $\mathbb{Z}_{\Pi}^*$. Conversely, if $k^{\lambda(\Pi)} \equiv 1 \pmod{\Pi}$ then, for all primes $p_i$ dividing $\Pi$, it follows that $k^{p_i - 1} \equiv 1 \pmod{p_i} \iff \gcd(k, p_i) = 1$, and thus $\gcd(k, \Pi) = 1 \iff k \in \mathbb{Z}_{\Pi}^*$. □

PROPOSITION.– Let $k, r$ be integers modulo $\Pi$ and assume that $\gcd(r, \Pi) = 1$. Then

$$[k + r(1 - k^{\lambda(\Pi)}) \bmod \Pi] \in \mathbb{Z}_{\Pi}^* \quad . \tag{7.1}$$

*Proof.* Let $\prod_i {p_i}^{\delta_i}$ denote the prime factorization of modulus $\Pi$. Define $\omega(k, r) \coloneqq [k + r(1 - k^{\lambda(\Pi)}) \bmod \Pi] \in \mathbb{Z}_{\Pi}$. Let $p_i$ be a prime factor of $\Pi$. Suppose that $p_i \mid k$ then $\omega(k, r) \equiv r \not\equiv 0 \pmod{p_i}$ since $\gcd(r, p_i)$ divides $\gcd(r, \Pi) = 1$. Suppose now that $p_i \nmid k$ then $k^{\lambda(\Pi)} \equiv 1 \pmod{p_i}$ and so $\omega(k, r) \equiv k \not\equiv 0 \pmod{p_i}$. Therefore for all primes $p_i \mid \Pi$, one has $\omega(k, r) \not\equiv 0 \pmod{p_i}$ and thus $\omega(k, r) \not\equiv 0 \pmod{{p_i}^{\delta_i}}$, which, invoking Chinese remaindering, concludes the proof. □

Benefiting from these facts, the unit generation method illustrated in Algorithm 8.1 can be devised.

**Algorithm 7.1**: Unit generation algorithm

**Input:** $\Pi \geq 2$ and $\lambda(\Pi)$
**Output:** a uniformly random unit $k \in \mathbb{Z}_{\Pi}^{*}$

1 Select $k \xleftarrow{\$} [1, \Pi)$ uniformly at random
2 Set $U \leftarrow (1 - k^{\lambda(\Pi)}) \bmod \Pi$
3 **if** $(U \neq 0)$ **then**
4 | Select $r \xleftarrow{\$} [1, \Pi)$ uniformly at random
5 | Set $k \leftarrow k + rU \pmod{\Pi}$
6 | Go to Step 2
7 **end if**
8 **return** $k$

As computed, the generation of units is self-correcting in the following sense: as soon as $k$ is co-prime with some factor of $\Pi$, it remains co-prime with this factor after the updating step $k \leftarrow k + rU \pmod{\Pi}$. This follows from Eq. (8.1). What happens in simple words is that, viewing $k$ as the vector of its residues $k \bmod {p_i}^{\delta_i}$ for all ${p_i}^{\delta_i} \mid \Pi$ (i.e., an RNS[2] representation of $k$ based on $\Pi$), non-invertible coordinates of $k$ are continuously re-randomized until invertibility is reached for all of them. This ensures that the output distribution is strictly uniform, provided that the random number generator outputs uniform integers over $[1, \Pi)$.

The above algorithm is particularly well suited to devices (e.g., smart cards) equipped with a co-processor to efficiently perform multiplications modulo $\Pi$. This usually requires $\Pi$ to lie within a certain range of supported values. For larger values of $\Pi$, the generation of units can be adapted as follows. $\Pi$ is written as a product of pairwise co-prime integers $\Pi_i \geq 2$,

$$\Pi = \prod_{i=1}^{w} \Pi_i \quad \text{with } \gcd(\Pi_i, \Pi_j) = 1 \text{ for } i \neq j \ .$$

Algorithm 8.1 is then run with every couple $(\Pi_i, \lambda(\Pi_i))$ as inputs. This yields a sequence of $M$ units $(k_1, \ldots, k_w)$ where $k_i \in \mathbb{Z}_{\Pi_i}^{*}$. There is no need to strictly apply

2. RNS stands for 'residue number system'; this system represents integers by their values modulo several pairwise co-prime integers.

Chinese remaindering to get a unit modulo $\Pi$ from $(k_1, \ldots, k_w)$ as long as the resulting value is invertible modulo $\Pi$. For example, one can compute iteratively

$$\begin{cases} K_0 = k_1 \\ K_j = \Pi_{j+1} K_{j-1} + \left(\prod_{i=1}^{j} \Pi_i\right) k_{j+1} \quad \text{for } 1 \leq j \leq w-1 \end{cases} \qquad [7.2]$$

and set $k \leftarrow K_{w-1} \bmod \Pi$. Letting $\epsilon_i \coloneqq \frac{\Pi}{\Pi_i} \in \mathbb{Z}$, it can be verified that the so-defined $k$ satisfies $k \equiv \epsilon_i \, k_i \pmod{\Pi_i}$ and thus $(k \bmod \Pi_i) \in \mathbb{Z}^*_{\Pi_i}$ since $\epsilon_i, k_i \in \mathbb{Z}^*_{\Pi_i}$. It can also be verified that $k$ remains uniform over $\mathbb{Z}^*_{\Pi}$, again assuming a uniform random number generator.

A different method to get units is to leverage the properties of quadratic residuosity. Given an odd prime $p_i$, an integer $-V$ is by definition a quadratic non-residue modulo $p_i$ if there exists no integer $t$ such that $-V \equiv t^2 \pmod{p_i}$. This implies that $t^2 + V$ is co-prime with $p_i$ for every integer $t$. Let $\Pi' = \prod_i {p_i}^{\delta_i}$ with $p_i$ prime, $p_i \neq 2$, and $\delta_i \geq 1$. With $(\Pi', V)$ precomputed such that $-V$ is a quadratic non-residue for every prime $p_i \mid \Pi'$, a unit $k \in \mathbb{Z}^*_{\Pi'}$ can be simply obtained as

$$k = (r^2 + V) \bmod \Pi' \quad \text{for some random } r \xleftarrow{\$} [0, \Pi') \ .$$

Such a unit $k$ is not uniform over $\mathbb{Z}^*_{\Pi}$. For each prime-power ${p_i}^{\delta_i}$, about half of the units in $\mathbb{Z}^*_{{p_i}^{\delta_i}}$ are covered. Hence, if $\Pi'$ is made of $w$ prime-power divisors ${p_i}^{\delta_i}$, about a subset of $\varphi(\Pi')/2^w$ units can be attained instead of the full set of $\varphi(\Pi')$ possible units. This can be mitigated by considering a product of $J$ independent units; namely,

$$k = \prod_{j=1}^{J} \left({r_j}^2 + V\right) \bmod \Pi' \quad \text{where } r_j \xleftarrow{\$} [0, \Pi') \ .$$

In practice, the value of $J$ is typically set to 6, which results in a min-entropy loss of at most $0.11$ bits.

## 7.4. Generation of Random Primes

This section describes a generic sieving algorithm for generating a random prime $q$ in some arbitrary interval $[q_{\text{min}}, q_{\text{max}}]$. The algorithm requires as inputs a smooth integer $\Pi = 2^{\delta}\Pi'$ (with $\delta \geq 0$ and $\Pi'$ odd) and a bound $c_{\text{max}} \geq 1$ on a counter that indicates the maximum number of re-uses of a fresh unit. Optionally, it further requires (i) the Carmichael's value $\lambda(\Pi)$ for generating (uniform) units modulo $\Pi$, and (ii) a pre-computed unit $U \in \mathbb{Z}^*_{\Pi}$ and/or a quadratic non-residue $-V$ modulo $\Pi'$ for

**Algorithm 7.2**: Prime generation algorithm in $[q_{\min}, q_{\max}]$

**Input:** $\Pi$, $c_{\max}$ [optionally: $\lambda(\Pi)$, $U$ and $V$]
**Output:** A random prime $q \in [q_{\min}, q_{\max}]$

Set $c \leftarrow 0$ and $M \leftarrow \left\lfloor \frac{q_{\max} - q_{\min}}{\Pi} \right\rfloor$
**repeat**
  **if** $(c = 0)$ **then**
    Generate $k \xleftarrow{\$} \mathbb{Z}_{\Pi}^{*}$
  **else**
    Sample $\upsilon \leftarrow \mathbb{Z}_{\Pi}^{*}$
    Update $k$ as $k \leftarrow k\upsilon \bmod \Pi$
  **end if**
  Increment $c$ as $c \leftarrow c + 1$
  **if** $(c \geq c_{\max})$ **then** $c \leftarrow 0$
  Set $L \leftarrow q_{\min} + \big((k - q_{\min}) \bmod \Pi\big)$
  Draw a random integer $m \xleftarrow{\$} [0, M]$ and set $q \leftarrow m\Pi + L$
**until** $(q \leq q_{\max})$ *and* $(\mathsf{T}(q) = 1)$
**return** $q$

quickly sampling units in $\mathbb{Z}_{\Pi}^{*}$. It also assumes some fast (pseudo-)primality testing function $\mathsf{T}$ which returns 1 when a candidate $q$ is found to be prime, and 0 otherwise.

The **repeat** loop involves the generation of a (uniform) unit modulo $\Pi$ when $c = 0$; this can for example be achieved using Algorithm 8.1. If $c \neq 0$, unit $k \in \mathbb{Z}_{\Pi}^{*}$ is updated as another unit $k \in \mathbb{Z}_{\Pi}^{*}$ (the product of two units is a unit). A prime candidate $q$ is next formed. Remark that by construction $q$ is co-prime with $\Pi$ since $q \equiv k \pmod{\Pi}$ and $k \in \mathbb{Z}_{\Pi}^{*}$. The process is iterated until $q \in [q_{\min}, q_{\max}]$ is declared prime.

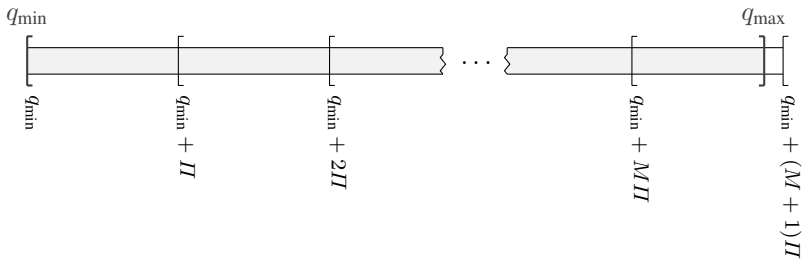


**Figure 7.1**: Output domain

Parameter $c_{\max}$ controls the distribution of the output primes. In the case of $c_{\max} = 1$, a fresh unit $k \in \mathbb{Z}_{\Pi}^*$ is generated for each tested prime candidate. If this unit is selected uniformly at random (e.g., with Algorithm 8.1), the prime $q$ returned by Algorithm 8.2 is uniformly distributed on the set of primes in $[q_{\min}, q_{\max}]$ (provided that $\mathsf{T}$ is correct in identifying $q$ as a prime).

Larger values for $c_{\max}$ enable various trade-offs running-time/uniformity for having units in $\mathbb{Z}_{\Pi}^*$. Several methods are available:

– The simplest way to update $k$ consists in predetermining a fixed unit $U \in \mathbb{Z}_{\Pi}^*$ and replacing $k$ with $k \leftarrow kU \bmod \Pi$ (i.e., $\upsilon = U$).

– Write $\Pi = 2^{\delta}\Pi'$ with $\delta \geq 0$ and $\Pi'$ odd. A random unit $u$ is first sampled in $\mathbb{Z}_{\Pi'}^*$ as $u \leftarrow (r^2 + V) \bmod \Pi'$ for a random integer $r \xleftarrow{\$} [0, \Pi')$. If $\delta = 0$ then $\upsilon = u$. Otherwise, $u \in \mathbb{Z}_{\Pi'}^*$ is extended as a unit $\upsilon$ in $\mathbb{Z}_{\Pi}^*$ as

$$\upsilon \leftarrow u + \big(2t + 1 - \mathrm{lsb}(u)\big)\Pi' \quad \text{for a random integer } t \xleftarrow{\$} [0, 2^{\delta-1} - 1] \ .$$

It can be checked that $\upsilon \in \mathbb{Z}_{\Pi}^*$ since $\upsilon \equiv u \pmod{\Pi'}$ and $u \in \mathbb{Z}_{\Pi'}^*$, $\upsilon \equiv 1 \pmod 2$, and $\upsilon \in [1, \Pi)$. Next, unit $k \in \mathbb{Z}_{\Pi}^*$ is updated as another unit $k \in \mathbb{Z}_{\Pi}^*$ as $k \leftarrow k\upsilon \bmod \Pi$.

The second method offers the advantage of being probabilistic. The output distribution for the resulting primes is expected to be statistically closer to the uniform distribution. Note also that the two methods can be combined.

### 7.4.1. *Probable primes*

The choice of function $\mathsf{T}$ dictates the type of primes that are generated. For probabilistic tests $\mathsf{T}$, numbers that pass the test are called *probable primes* or *pseudo-primes* as there is a non-zero probability that a composite number is falsely classified as prime. An example of such a function $\mathsf{T}$ is Fermat's test: $\mathsf{T}(q) = 1$ if $a^{q-1} \equiv 1 \pmod q$ for some random base $a > 1$. Miller–Rabin is usually preferred as it is more discriminative. The Miller–Rabin test writes (odd) prime candidate $q$ as $q = 2^D q' + 1$ with $q'$ odd and returns $\mathsf{T}(q) = 1$ if for some random base $a > 1$, it holds that

1) $a^{q'} \equiv 1 \pmod q$, or

2) $a^{2^d q'} \equiv -1 \pmod q$ for some $0 \leq d < D$.

Let $P(n, t)$ denote the probability that an $n$-bit odd integer is composite if it successfully passes $t$ iterations of the Miller–Rabin test. It can be shown that $P(n, 1) \leq n^2 4^{2-\sqrt{n}}$ for all $n \geq 2$, and $P(n, t) \leq 4^{1-t} P(n, 1)/(1 - P(n, 1))$ for every $n \geq 2$, $t \geq 2$. Stronger estimates for $P(n, t)$ are provided in the next table.

Hence, by defining function $\mathsf{T}$ as the repetition of Miller–Rabin with $t$ random bases $a$, an odd composite $n$-bit integer $q$ will be incorrectly declared prime with probability at most $P(n, t)$. From Table 8.1, it turns out that $P(n, t)$ is already $< 2^{-80}$ with $t = 2$ Miller–Rabin trials for prime candidates of bit-length $n \geq 600$.

| $n\backslash t$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 100 | 5 | 14 | 20 | 25 | 29 | 33 | 36 | 39 | 41 | 44 |
| 150 | 8 | 20 | 28 | 34 | 39 | 43 | 47 | 51 | 54 | 57 |
| 200 | 11 | 25 | 34 | 41 | 47 | 52 | 57 | 61 | 65 | 69 |
| 250 | 14 | 29 | 39 | 47 | 54 | 60 | 65 | 70 | 75 | 79 |
| 300 | 19 | 33 | 44 | 53 | 60 | 67 | 73 | 78 | **83** | 88 |
| 350 | 28 | 38 | 48 | 58 | 66 | 73 | **80** | 86 | 91 | 97 |
| 400 | 37 | 46 | 55 | 63 | 72 | **80** | 87 | 93 | 99 | 105 |
| 450 | 46 | 54 | 62 | 70 | 78 | **85** | 93 | 100 | 106 | 112 |
| 500 | 56 | 63 | 70 | 78 | **85** | 92 | 99 | 106 | 113 | 119 |
| 550 | 65 | 72 | 79 | **86** | 93 | 100 | 107 | 113 | 119 | 126 |
| 600 | 75 | **82** | 88 | 95 | 102 | 108 | 115 | 121 | 127 | 133 |

**Table 7.1**: Lower bounds for $-\log_2 P(n,t)$

The Miller–Rabin test can also be coupled with the Lucas test.

### 7.4.2. *Provable primes*

Deterministic tests T guarantee that the tested number is prime. They are however not truly practical. An alternative is to rely on methods derived from Pocklington's criterion. Unlike the Fermat's or Miller–Rabin tests, those methods provide sufficient conditions for primality. This is exemplified by the next proposition.

PROPOSITION.– Let $p > 2$ be an odd prime and let $q = 2rp + 1$ for some positive integer $r \leq p^2 + p + 1$. If there exists an integer $a$ such that

$$(i)\ a^{q-1} \equiv 1 \pmod{q} \text{ and } a^{2r} \not\equiv 1 \pmod{q}$$
$$(ii)\ r = up + s \text{ for some } 1 \leq s < p \text{ and } u \text{ odd}$$

then $q$ is prime.

*Proof.* Suppose that $q = 2rp + 1$ is composite. Hence, it must have an odd prime divisor $q_0$ and so can be written as $q = q_0 q_1$ where $q_1 = q/q_0$ is odd. Assume that $a^{q-1} \equiv 1 \pmod{q}$ and $a^{2r} \not\equiv 1 \pmod{q}$ for some integer $a$. Define $b = a^{2r} \bmod q_0$. From $q_0 \mid q$, this yields $b^p \equiv 1 \pmod{q_0}$ with $b \not\equiv 1 \pmod{q_0}$. Further, $q_0$ being prime, it also holds that $b^{q_0-1} \equiv 1 \pmod{q_0}$. Lagrange's theorem and the primality of $p$ imply that $p < q_0 - 1$ and in turn that $p \mid (q_0 - 1)$; see Exerc. 4. Therefore, prime $q_0$ must be of the form $q_0 = 2xp + 1$ for some integer $x \geq 1$. As a result, cofactor $q_1$ satisfies $q_1 = q \bmod (q_0 - 1) = (2rp + 1) \bmod 2xp = 2(r \bmod x)p + 1$. Letting $y = r \bmod x$, the product $q = q_0 q_1$ then leads to $q = 2(2xyp + x + y)p + 1$

and so $r = (2xy)p + (x + y)$, which contradicts the parity of $u := 2xy$. Note that $s := x + y$ verifies $1 \leq s < p$ since $x + y = x + (r \bmod x) < 2x = (q_0 - 1)/p$ and $q_0 \leq \sqrt{q} \leq \sqrt{2(p^2 + p + 1)p + 1} < p^2$ for $p > 2$. □

This proposition suggests a constructive method for generating (non-uniform) provable primes. One starts with a prime $p$ and—provided that Conditions $(i)$ and $(ii)$ are met—obtains a prime $q = 2(up + s)p + 1$ that is about 2 to 3 times longer. Iterating the process eventually leads to a prime of the desired length. The initial prime can be chosen as any integer in the range $[2, 2^{32})$ that successfully passes the Miller–Rabin test with the three bases $(2, 3, 61)$; all those integers are known to be prime.

In order to increase the likelihood that $q = 2rp + 1$ with $r = up + s$ verifies Conditions $(i)$ and $(ii)$ and so is actually a prime, it is constructed in a way to be automatically co-prime with many small primes. Specifically, let $\Pi' \leq p^2 + p + 1$ be an odd smooth integer, with $p \nmid \Pi'$. If $r \equiv k - \frac{1}{2p} \pmod{\Pi'}$ for some $k \xleftarrow{\$} \mathbb{Z}^*_{\Pi'}$, it follows that $q \bmod 2 = 1$ and $q \equiv 2pk \pmod{\Pi'}$, and thus $\gcd(q, 2\Pi') = \gcd(q, \Pi') = 1$ since $2pk \in \mathbb{Z}^*_{\Pi'}$.

## 7.5. RSA Key Generation

An RSA modulus $N = pq$ is the product of two large prime numbers $p$ and $q$. If $\ell$ denotes the bit-length of $N$ then, for some $1 < \ell_0 < \ell$, $p$ must lie in the range $\left[\left\lceil 2^{\ell - \ell_0 - \frac{1}{2}} \right\rceil, 2^{\ell - \ell_0}\right]$ and $q$ in the range $\left[\left\lceil 2^{\ell_0 - \frac{1}{2}} \right\rceil, 2^{\ell_0}\right]$ so that $2^{\ell - 1} < N = pq < 2^{\ell}$. For security reasons, so-called balanced moduli are generally preferred, which means $\ell = 2\ell_0$. This corresponds to primes $p$ and $q$ being drawn at random in the interval $[q_{\text{min}}, q_{\text{max}}]$ where $q_{\text{min}} = \left\lceil 2^{\ell_0 - \frac{1}{2}} \right\rceil$ and $q_{\text{max}} = 2^{\ell_0}$. Furthermore, for an RSA modulus $N = pq$, the primes $p$ and $q$ being generated must verify the condition $\gcd(p - 1, e) = \gcd(q - 1, e) = 1$ for a selected public exponent $e$. Matching private exponent $d$ is given by an integer that is congruent to $e^{-1}$ modulo $\operatorname{lcm}(p - 1, q - 1)$; in practice, $d$ is often set to $d \leftarrow e^{-1} \bmod (p - 1)(q - 1)$.

NOTE.– A reminiscence of history is the use of so-called safe, strong, or X9.31 RSA primes. The reason of using such primes was to prevent certain classes of attacks. In particular, they were introduced to better resist cycling attacks and the $(p - 1)$ and $(p + 1)$ factoring attacks. Cyclic attacks were shown to have a negligible chance to succeed, whatever the form of the RSA primes. The $(p - 1)$ and $(p + 1)$ factoring attacks are now obsolete owing to new factorization algorithms—in particular, the elliptic curve method (ECM). It is therefore recommended to generate random RSA primes rather than special primes.

By construction, the prime generation algorithms of Section 8.4 output a prime candidate $q \in [q_{\text{min}}, q_{\text{max}}]$ such that $q \equiv k \pmod{\Pi}$ for some random unit $k \in \mathbb{Z}^*_{\Pi}$.

The product of primes, $\Pi = \prod_i p_i$, is chosen so as to minimize the ratio $\varphi(\Pi)/\Pi$ subject to $\Pi < q_{\max} - q_{\min}$. Euler's totient function $\varphi(\Pi)$ represents the group order (i.e., the number of elements) of $\mathbb{Z}_\Pi^*$. The minimality of $\varphi(\Pi)/\Pi$ lowers the expected number of trials before a prime is identified. This is achieved by ensuring that $\Pi$ contains a maximum number of distinct primes and that these primes are as small as possible. For example, for the generation of 1024-bit RSA primes, one can select $\Pi = 2 \cdot 3 \cdot 5 \cdots 739$ as the product of the first 131 primes, namely

$$\begin{aligned}\Pi_{1024} = \texttt{0x}\,&\texttt{0590 bff0 e1e4 97d0 1ec5 6374 d841 7ae5}\\ &\texttt{706f dfbe 2424 0db0 fb6a d6fa d3f4 9804}\\ &\texttt{3820 f879 440d fd1f 4e10 1dea eddf f905}\\ &\texttt{f362 1eae a0ca d6ef 2962 d5fa e132 dd23}\\ &\texttt{5ded c15a 2bd2 360e 3593 649b 3164 675b}\\ &\texttt{0ddc 0aaa 31cb 1cac c71d e317 58b8 e996}\\ &\texttt{6b77 6dcb b5c4 f07b a381 9cfb d89f 8e1c}\\ &\texttt{a30d a823 be6c 6d1e 0c35 46c0 23e6 02f2}\end{aligned}$$

Companion parameters are Carmichael's value $\lambda(\Pi_{1024})$ and quadratic non-residue $-V_{1024}$ modulo $\Pi_{1024}/2$. One has $\lambda(\Pi_{1024}) = \texttt{0x}\,\texttt{0009 220e 37a8 2cbb 6007 a9de e07d e852 b1fd 11d7 5946 8826 4f7f 40e7 1355 f33b 7ebf c100}$. Observe that the bit-length of $\lambda(\Pi_{1024})$ is much smaller than that of $\Pi_{1024}$: $|\lambda(\Pi_{1024})|_2 = 276$ while $|\Pi_{1024}|_2 = 1019$. For $V_{1024}$, one can take

$$\begin{aligned}V_{1024} = \texttt{0x}\,&\texttt{005b fdb1 a66b f64b f262 42fc b803 1844}\\ &\texttt{ca3a 2182 ad42 294e 294d 40d7 61e8 552f}\\ &\texttt{2051 4fae 12e2 e3ae 6e1d e402 4b68 4d98}\\ &\texttt{5548 1fd9 c208 fd89 839c ff93 37a3 f8f9}\\ &\texttt{2c16 6dff d1a7 ce2f 3b14 2ca0 8121 68f2}\\ &\texttt{aaa6 e720 a340 2108 7bb9 71a3 5edc 796d}\\ &\texttt{ed2f ef6d 1651 a9bc 6a23 4693 254b 7b2f}\\ &\texttt{1cd1 2053 c4e6 6755 c506 8c07 479c 3310}\end{aligned}$$

The private operation in RSA (i.e., decryption or signature generation) can be sped up through Chinese remaindering: the private operation is carried out modulo each prime factor of modulus $N$ and these partial results are then recombined. In more detail, if $N = pq$ and $d$ denotes the private exponent, one defines

$$d_p = d \bmod (p-1)\,, \quad d_q = d \bmod (q-1)\,, \quad i_q = q^{-1} \bmod p$$

and, given $C$, computes $C^d \bmod N$ as $\mathrm{CRT}(x_p, x_q) := x_q + q\big[i_q(x_p - x_q) \bmod p\big]$ from $x_p \leftarrow C^{d_p} \bmod p$ and $x_q \leftarrow C^{d_q} \bmod q$. This mode of operation is referred

to as *CRT mode* and the private parameters are $\{p, q, d_p, d_q, i_q\}$. Compared to the standard (i.e., non-CRT) mode, the computation time is expected to be quartered.

It remains to demonstrate (i) how to ensure that primes $p$ and $q$ verify the additional constraint $e \nmid (p-1)$ and $e \nmid (q-1)$; (ii) how to get private key $d$; and (iii) how to get CRT parameters $(d_p, d_q, i_q)$. *The rest of this section assumes that public exponent $e$ is a (small) prime as is usually required in the vast majority of embedded implementations.* The most frequently used public exponent is $e = 2^{16} + 1$; other popular exponents are $e = 3$ and $e = 17$.

The conditions $e \nmid (p-1)$ and $e \nmid (q-1)$ when $e$ is a small prime translate into $p, q \not\equiv 1 \pmod{e}$. As a reminder, prime $p$ is constructed in a way of being congruent to some unit $k$ modulo $\Pi$; and similarly for prime $q$. There are two cases to consider:

1) $e \mid \Pi$: In this case, the candidate unit $k$ is initialized as $k \leftarrow k_0 + er \pmod{\Pi}$ with $k_0 \xleftarrow{\$} [2, \ldots, e-1]$ so that $k \equiv k_0 \not\equiv 0, 1 \pmod{e}$. Doing so, the output of Algorithm 8.1 is a unit $k \in \mathbb{Z}_\Pi^*$ such that $k \not\equiv 1 \pmod{e}$.

2) $e \nmid \Pi$: A verification step has then to be explicitly added on the prime candidates; namely, are $p, q \not\equiv 1 \pmod{e}$? When applicable, this verification can be done before or after (pseudo-)primality test $\mathsf{T}$ is applied.

Given public exponent $e$, corresponding private exponent $d$ can be set as any value that is congruent to $e^{-1}$ modulo $\lambda(N)$. In order to avoid computing $\gcd(p-1, q-1)$, $d$ is usually defined as $d = e^{-1} \bmod \varphi(N)$ where $\varphi(N) = (p-1)(q-1)$—observe that such a $d \equiv e^{-1} \pmod{\lambda(N)}$ since $\varphi(N) = \gcd(p-1, q-1) \cdot \lambda(N) \propto \lambda(N)$. Modular inverses can be obtained via Euclid's algorithm, which essentially amounts to compute a GCD. A method better suited to embedded platforms relies on Arazi's inversion formula. It enables expressing the inverse of $e$ modulo $f$ as a function of the inverse of $f$ modulo $e$. This is stated in the next proposition.

PROPOSITION.– Let $e$ and $f$ be two positive integers. If $\gcd(e, f) = 1$ then

$$e^{-1} \bmod f = \frac{1 + f(-f^{-1} \bmod e)}{e} \quad . \qquad [7.3]$$

*Proof.* Define $U = e(e^{-1} \bmod f) + f(f^{-1} \bmod e)$. Since $U \equiv 1 \pmod{e}$ and $U \equiv 1 \pmod{f}$, it follows that $U \equiv 1 \pmod{ef}$. Hence, noting that $1 < e + f \leq U < 2ef$, this implies that $U = 1 + ef$ or, equivalently, that $e^{-1} \bmod f = \frac{1}{e}\big[(1 + ef) - f(f^{-1} \bmod e)\big] = \frac{1}{e}\big[1 + f(-f^{-1} \bmod e)\big]$, as desired. □

Taking $f := (p-1)(q-1)$, a valid value for private exponent $d$ is therefore given by $d = \frac{1+f(-f^{e-2} \bmod e)}{e}$. Note that this requires $e$ being prime. From $d$, private

CRT exponents $d_p$ and $d_q$ are then directly obtained as $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$. Since $p$ is prime, CRT parameter $i_q$ can be computed by an application of Fermat Little Theorem as $i_q = q^{p-2} \bmod p$.

## 7.6. Exercices

1) Let $w$ pairwise co-prime integers $\Pi_1, \ldots, \Pi_w$ with $\Pi_i \geq 2$ and let $\Pi = \prod_{i=1}^{w} \Pi_i$. Let also integers $\epsilon_i = \Pi/\Pi_i$. Given $w$ integers $k_1, \ldots, k_w$ with $k_i \in \mathbb{Z}^*_{\Pi_i}$ (viewed as elements in $[1, \Pi_i - 1]$), define

$$\begin{cases} K_0 = k_1 \\ K_j = \Pi_{j+1} K_{j-1} + \left(\prod_{i=1}^{j} \Pi_i\right) k_{j+1} \quad \text{for } 1 \leq j \leq w-1 \end{cases}.$$

Prove that $\gcd(K_{w-1}, \Pi) = 1$.

2) Factoring-based constructs typically make use of RSA moduli $N = pq$ with primes $p$ and $q$ that are congruent to 3 modulo 4. This is for example the case in the Fiat–Shamir identification protocol. Supposing that candidate prime $p$ (resp. $q$) always remains co-prime with some unit modulo $\Pi$, how to tweak the unit generation algorithm (Algorithm 8.1) so that the condition $p \equiv 3 \pmod 4$ (resp. $q \equiv 3 \pmod 4$) is automatically satisfied? Can this be extended to support Rabin–Williams moduli, that is, moduli $N = pq$ with $p \equiv 3 \pmod 8$ and $q \equiv 7 \pmod 8$?

3) A DSA prime is an $\ell$-bit prime $q$ of the form $q = 1 + pr$ where $p$ is also a prime. Given $p$ and $\ell$, the goal is to find an integer $r$ such that $1 + pr$ is a prime in $[2^\ell, 2^\ell - 1]$. Let $\Pi$ denote a product of prime numbers with $p \nmid \Pi$. Remark that if $r \equiv -1/p + k \pmod \Pi$ for some unit $k \in \mathbb{Z}^*_\Pi$ then $\gcd(1 + pr, \Pi) = 1$. Use this observation to design an efficient generator for DSA primes.

4) The order of an element $b \in \mathbb{Z}^*_q$ is the smallest positive integer $n$ such that $b^n \equiv 1 \pmod q$. The order of a group is the number of its elements. Lagrange's theorem says that the order of an element always divides the order of its group. Let $q$ be a prime and $b \in \mathbb{Z}^*_q$ with $b \not\equiv 1 \pmod q$. Prove that $b^p \equiv 1 \pmod q$ for some prime $p$ implies that $p \mid (q-1)$.

5) Check that for each prime $p_i \neq 2$ dividing $\Pi_{1024}$ (see Section 8.5), the value of $V_{1024}$ is such that $V_{1024} \bmod p_i \in \{1, 2, 5, 19\}$. Deduce a more compact representation for the pair of parameters $(\Pi_{1024}, V_{1024})$ and apply Exercise 1 for generating units modulo $\Pi_{1024}$.

6) Let $N = pq$ be an RSA modulus and let $(e, d)$ denote the matching pair of public/private RSA exponents. Let also $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$. Prove that $C^{d-1} \equiv p y_q + q y_p \pmod N$ where $y_q = \left(p(Cp)^{e-1}\right)^{q-1-d_q} \bmod q$ and $y_p = \left(q(Cq)^{e-1}\right)^{p-1-d_p} \bmod p$. Use this relation to derive a formula for computing $C^d \bmod N$ from CRT parameters $\{p, q, d_p, d_q\}$ (i.e., without using $i_q$). Estimate the incurred overhead compared to the usual CRT recombination when the public exponent is $e = 2^{16} + 1$.

7) Given an odd integer $D$, define the recurrence relation

$$\begin{cases} x_0 = 1 \\ x_n = x_{n-1}(2 - D\,x_{n-1}) \bmod 2^{2^n} \quad \text{for } n \geq 1 \end{cases}.$$

Show that $x_n = D^{-1} \bmod 2^{2^n}$. Explain how this can be used for quickly evaluating the integer division in Arazi's inversion formula (i.e., the integer division by $e$ in Eq. (8.3)).

8) Find a way to reconstruct private exponent $d$ from $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$ without computing $\gcd(p-1, q-1)$.

## 7.7. Bibliography

[1] Albrecht, M. R., Massimo, J., Paterson, K. G., Somorovsky, J. (2018), Prime and prejudice: Primality testing under adversarial conditions, *in* D. Lie, M. Mannan, M. Backes, X. Wang, (eds), ACM CCS 2018: 25th Conference on Computer and Communications Security, ACM Press, pp. 281–298.

[2] Aldaya, A. C., García, C. P., Tapia, L. M. A., Brumley, B. B. (2019), Cache-timing attacks on RSA key generation, *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(4), 213–242.

[3] ANSI X9.80-2020 (2020), *Prime Number Generation, Primality Testing, and Primality Certificates*, American National Standards Institute. American National Standard for Financial Services.

[4] Atkin, A. O. L., Morain, F. (1993), Elliptic curves and primality proving, *Mathematics of Computation*, 61(203), 29–68.

[5] Baillie, R., Wagstaff Jr., S. S. (1980), Lucas pseudoprimes, *Mathematics of Computation*, 35(152), 1391–1417.

[6] Bauer, A., Jaulmes, É., Lomné, V., Prouff, E., Roche, T. (2014), Side-channel attack against RSA key generation algorithms, *in* L. Batina, M. Robshaw, (eds), Cryptographic Hardware and Embedded Systems – CHES 2014, vol. 8731 of *Lecture Notes in Computer Science*, Springer, Heidelberg, pp. 223–241.

[7] Beauchemin, P., Brassard, G., Crépeau, C., Goutier, C., Pomerance, C. (1988), The generation of random numbers that are probably prime, *Journal of Cryptology*, 1(1), 53–64.

[8] Bellare, M., Rogaway, P. (1993), Random oracles are practical: A paradigm for designing efficient protocols, *in* D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, V. Ashby, (eds), ACM CCS 93: 1st Conference on Computer and Communications Security, ACM Press, pp. 62–73.

[9] Bellare, M., Rogaway, P. (1995), Optimal asymmetric encryption, *in* A. De Santis, (ed.), Advances in Cryptology – EUROCRYPT'94, vol. 950 of *Lecture Notes in Computer Science*, Springer, Heidelberg, pp. 92–111.

[10] Bellare, M., Rogaway, P. (1996), The exact security of digital signatures: How to sign with RSA and Rabin, *in* U. M. Maurer, (ed.), Advances in Cryptology – EUROCRYPT'96, vol. 1070 of *Lecture Notes in Computer Science*, Springer, Heidelberg, pp. 399–416.

[11] Bosma, W., van der Hulst, M.-P. (1990), Faster primality testing (extended abstract) (rump session), *in* J.-J. Quisquater, J. Vandewalle, (eds), Advances in Cryptology – EUROCRYPT'89, vol. 434 of *Lecture Notes in Computer Science*, Springer, Heidelberg, pp. 652–656.

[12] Brandt, J., Damgård, I. (1993), On generation of probable primes by incremental search, *in* E. F. Brickell, (ed.), Advances in Cryptology – CRYPTO'92, vol. 740 of *Lecture Notes in Computer Science*, Springer, Heidelberg, pp. 358–370.

[13] Brandt, J., Damgård, I., Landrock, P. (1993), Speeding up prime number generation, *in* H. Imai, R. L. Rivest, T. Matsumoto, (eds), Advances in Cryptology – ASIACRYPT'91, vol. 739 of *Lecture Notes in Computer Science*, Springer, Heidelberg, pp. 440–449.

[14] Brillhart, J., Lehmer, D. H., Selfridge, J. L., Tuckerman, B., Wagstaff Jr., S. S. (2002), *Factorizations of* $b^n \pm 1$, $b = 2, 3, 5, 6, 7, 10, 11, 12$ *Up to High Powers*, vol. 22 of *Contemporary Mathematics*, 3rd edition, American Mathematical Society, Providence, R.I.

[15] Clavier, C., Coron, J.-S. (2007), On the implementation of a fast prime generation algorithm, *in* P. Paillier, I. Verbauwhede, (eds), Cryptographic Hardware and Embedded Systems – CHES 2007, vol. 4727 of *Lecture Notes in Computer Science*, Springer, Heidelberg, pp. 443–449.

[16] Clavier, C., Feix, B., Thierry, L., Paillier, P. (2012), Generating provable primes efficiently on embedded devices, *in* M. Fischlin, J. Buchmann, M. Manulis, (eds), PKC 2012: 15th International Conference on Theory and Practice of Public Key Cryptography, vol. 7293 of *Lecture Notes in Computer Science*, Springer, Heidelberg, pp. 372–389.

[17] Crandall, R., Pomerance, C. B. (2005), *Prime Numbers: A Computational Perspective*, 2nd edition, Springer.

[18] Damgård, I., Landrock, P., Pomerance, C. (1993), Average case error estimates for the strong probable prime test, *Mathematics of Computation*, 61(203), 177–194.

[19] Dussé, S. R., Kaliski Jr., B. S. (1991), A cryptographic library for the Motorola DSP56000, *in* I. Damgård, (ed.), Advances in Cryptology – EUROCRYPT'90, vol. 473 of *Lecture Notes in Computer Science*, Springer, Heidelberg, pp. 230–244.

[20] Finke, T., Gebhardt, M., Schindler, W. (2009), A new side-channel attack on RSA prime generation, *in* C. Clavier, K. Gaj, (eds), Cryptographic Hardware and Embedded Systems – CHES 2009, vol. 5747 of *Lecture Notes in Computer Science*, Springer, Heidelberg, pp. 141–155.

[21] FIPS186-4 (2013), *Digital Signature Standard (DSS)*, American National Standards Institute. Federal Information Processing Standards Publication.

[22] Fischer, W., Seifert, J.-P. (2002), Note on fast computation of secret RSA exponents, *in* L. M. Batten, J. Seberry, (eds), ACISP 02: 7th Australasian Conference on Information Security and Privacy, vol. 2384 of *Lecture Notes in Computer Science*, Springer, Heidelberg, pp. 136–143.

[23] Fouque, P.-A., Tibouchi, M. (2014), Close to uniform prime number generation with fewer random bits, *in* J. Esparza, P. Fraigniaud, T. Husfeldt, E. Koutsoupias, (eds), ICALP 2014: 41st International Colloquium on Automata, Languages and Programming, Part I, vol. 8572 of *Lecture Notes in Computer Science*, Springer, Heidelberg, pp. 991–1002.

[24] Hamburg, M., Tunstall, M., Xiao, Q. (2021), Improvements to RSA key generation and CRT on embedded devices, *in* K. G. Paterson, (ed.), Topics in Cryptology – CT-RSA 2021, vol. 12704 of *Lecture Notes in Computer Science*, Springer, Heidelberg, pp. 633–656.

[25] Heninger, N., Durumeric, Z., Wustrow, E., Halderman, J. A. (2012), Mining your Ps and Qs: Detection of widespread weak keys in network devices, *in* T. Kohno, (ed.), USENIX Security 2012: 21st USENIX Security Symposium, USENIX Association, pp. 205–220.

[26] IEEE Std 1363-2000 (2000), *IEEE Standard Specifications for Public-Key Cryptography*, IEEE Computer Society.

[27] ISO/IEC 18031:2011 (2011), *Information Technology – Security Techniques – Random Bit Generation*, International Organization for Standardization.

[28] ISO/IEC 18032:2020 (2020), *Information Security – Prime Number Generation*, International Organization for Standardization.

[29] Jaeschk, G. (1993), On strong pseudoprimes to several bases, *Mathematics of Computation*, 61(204), 915–926.

[30] Joye, M., Paillier, P. (2003), GCD-free algorithms for computing modular inverses, *in* C. D. Walter, Ç. K. Koç, C. Paar, (eds), Cryptographic Hardware and Embedded Systems – CHES 2003, vol. 2779 of *Lecture Notes in Computer Science*, Springer, Heidelberg, pp. 243–253.

[31] Joye, M., Paillier, P. (2006), Fast generation of prime numbers on portable devices: An update, *in* L. Goubin, M. Matsui, (eds), Cryptographic Hardware and Embedded Systems – CHES 2006, vol. 4249 of *Lecture Notes in Computer Science*, Springer, Heidelberg, pp. 160–173.

[32] Joye, M., Paillier, P., Vaudenay, S. (2000), Efficient generation of prime numbers, *in* Çetin Kaya. Koç, C. Paar, (eds), Cryptographic Hardware and Embedded Systems – CHES 2000, vol. 1965 of *Lecture Notes in Computer Science*, Springer, Heidelberg, pp. 340–354.

[33] Landrock, P. (1999), Primality tests and use of primes in public key systems, *in* I. Damgård, (ed.), Lectures on Data Security, vol. 1561 of *Lecture Notes in Computer Science*, Springer, Heidelberg, pp. 127–133.

[34] Lenstra, A. K., Hughes, J. P., Augier, M., Bos, J. W., Kleinjung, T., Wachter, C. (2012), Public keys, *in* R. Safavi-Naini, R. Canetti, (eds), Advances in Cryptology – CRYPTO 2012, vol. 7417 of *Lecture Notes in Computer Science*, Springer, Heidelberg, pp. 626–642.

[35] Maurer, U. M. (1995), Fast generation of prime numbers and secure public-key cryptographic parameters, *Journal of Cryptology*, 8(3), 123–155.

[36] Mihailescu, P. (1994), Fast generation of provable primes using search in arithmetic progressions, *in* Y. Desmedt, (ed.), Advances in Cryptology – CRYPTO'94, vol. 839 of *Lecture Notes in Computer Science*, Springer, Heidelberg, pp. 282–293.

[37] Nemec, M., Sýs, M., Svenda, P., Klinec, D., Matyas, V. (2017), The return of Coppersmith's attack: Practical factorization of widely used RSA moduli, *in* B. M. Thuraisingham, D. Evans, T. Malkin, D. Xu, (eds), ACM CCS 2017: 24th Conference on Computer and Communications Security, ACM Press, pp. 1631–1648.

[38] Pocklington, H. C. (1914), The determination of the prime or composite nature of large numbers by Fermat's theorem, *Proc. of the Cambridge Philosophical Society*, 18, 29–30.

[39] Rabin, M. O. (1980), Probabilistic algorithm for testing primality, *Journal of Number Theory*, 12(1), 128–138.

[40] Rivest, R. L., Shamir, A., Adleman, L. M. (1978), A method for obtaining digital signatures and public-key cryptosystems, *Communications of the Association for Computing Machinery*, 21(2), 120–126.

[41] Rivest, R., Silverman, R. (2001), 'Are strong primes needed for RSA', Cryptology ePrint Archive, Report 2001/007. https://ia.cr/2001/007.

[42] Weiser, S., Spreitzer, R., Bodner, L. (2018), Single trace attack against RSA key generation in Intel SGX SSL, *in* J. Kim, G.-J. Ahn, S. Kim, Y. Kim, J. López, T. Kim, (eds), ASIACCS 18: 13th ACM Symposium on Information, Computer and Communications Security, ACM Press, pp. 575–586.

[43] Young, A. (2004), Mitigating insider threats to RSA key generation, *CryptoBytes*, 7(1), 1–15.

## NOTES AND FURTHER REFERENCES

**Section 8.1** An excellent general reference to prime numbers is [17]. The RSA cryptosystem [40] is named after its inventors Rivest, Shamir, and Adleman. Widely used RSA padding functions are OAEP [9] for message encryption and FDH [8] or PSS [10] for digital signatures. The naïve prime generator is examined in [12]. Its extension using a set of a small primes is described in [32]. Studies of RSA keys found in the wild were conducted in 2012 in two independent works: [25] and [34]. Setting the length of the initial seed to twice the security length associated with the RSA modulus is a NIST recommendation [21, Appendix B.3.2]. Methods for generating numbers from a sequence of random bits are provided in ISO/IEC 18031 [27]. Discussions regarding side-channel attacks can be found in [2, 6, 15, 20, 42].

**Section 8.2** ANSI standard X9.80 [3] is a very good reference on the different (pseudo-)primality tests used in public-key cryptography, including for the RSA cryptosystem. An analysis of their strength under adversarial conditions is provided in [1]. Pocklington's test appears in [38] and its elliptic curve variant in [4]. The Jacobi sum test is described in [11].

**Section 8.3** A CRT sieve for sampling in $\mathbb{Z}_{\Pi}^{*}$ with $\Pi = \prod_{i=1}^{L} {p_i}^{\delta_i}$ using the Chinese Remainder Theorem is given in [32, § 4.1]. It however requires pre-computing and storing a large sequence of constants $\{\theta_i\}_{1 \leq i \leq L}$ where $\theta_i \equiv 1 \pmod{{p_i}^{\delta_i}}$ and $\theta_i \equiv 0 \pmod{{p_j}^{\delta_j}}$ for $j \neq i$. The generation of units as per Algorithm 8.1 is presented in [31, Fig. 2]. Advantageously, it only takes $\Pi$ and $\lambda(\Pi)$ as pre-computed inputs. The method building units from a quadratic non-residue $-V$ modulo $\Pi$ is presented in [24, § 2.3]. The pre-computed inputs in this case are $\Pi$ and $V$.

**Section 8.4** The generation of prime numbers is covered in part in several cryptographic standards, including ISO/IEC 18032 [28], ANSI X9.80 [3], and IEEE Std 1363 [26]. The general presentation of Algorithm 8.2 is adapted from [31, Fig. 2]. The main difference is the full coverage of the interval $[q_{\min}, q_{\max}]$ for prime candidates $q$. Rejection sampling is applied in the case $q > q_{\max}$. As presented, Algorithm 8.2 also encompasses the prime generation algorithm given in [24, Alg. 5]. See also [23] for a discussion on the output distribution. The failure probability of the Miller–Rabin test [39] is discussed in [7, 33]. Explicit functions that bound $P(n, t)$ are provided in [18]. Table 8.1 is reproduced from [18, Table 2]. The Lucas test is presented in [5]. The main proposition in Section 8.4.2 is a slight adaptation from [33, Theorem 1]. It simplifies a special case of [14, Theorem 11, Sect. III.B.2] used in [16] that requires an extra GCD computation. The corresponding prime generation methods reduce the number of iterations compared to earlier algorithms based on Pocklington's criterion [35, 36]; see also [13, Sect. 3]. Choices of Miller–Rabin bases that are necessary for proving primality up to a certain bound are given in [29, Sect. 5].

**Section 8.5** Good sources of practical information regarding the generation of RSA parameters are [31, 32]. Arguments against the use of special RSA primes are clarified in [41]. Lattice-based attacks against RSA keys using random primes with too few entropy are reported in [37]. A frequency analysis of a large collection of public RSA exponents appears in [34]. Insider attacks against RSA

key generation and mitigation measures are surveyed in [43]. Arazi's formula is named after Arazi who was the first to implement fast modular inversions of RSA exponents on a crypto-coprocessor. Its application to (small) prime exponents $e$ is described in [22]. Generalizations to arbitrary exponents $e$ and various implementation tricks are detailed in [30].

**Section 8.6** 1) Constructing a unit modulo $\Pi = \prod \Pi_i$ for pairwise co-prime moduli $\Pi_i$ can be done easily from units modulo each $\Pi_i$ using the Chinese Remainder Theorem (CRT). The improved method that do not require pre-computing CRT constants is described in [24, Algorithm 3].

2) This is an easy adaptation of Algorithm 8.1. The trick is to include 4 as a factor of $\Pi$. Variable $k$ is initialized with $k \leftarrow 3 + 4r$ for some $r \xleftarrow{\$} [0, \Pi/4)$. Note that $k \in [1, \Pi)$ and $k \equiv 3 \pmod 4$. Rabin–Williams moduli are supported analogously by including 8 as a factor of $\Pi$ and initializing $k$ accordingly.

3) An algorithm for generating DSA primes from units modulo $\Pi$ is presented in [32, § 7.1]. Algorithms for the generation of safe, strong, and X9.31 RSA primes are also presented therein; see also [31, § 4.2].

4) For a prime $q$, the order of $\mathbb{Z}_q^*$ is $q-1$. Let $n$ denote the order of $b \in \mathbb{Z}_q^*$, $b \not\equiv 1 \pmod q$. Lagrange's theorem implies that $n \mid (q-1)$ and $n \mid p$ since $b^n \equiv 1 \pmod q$. As $n$ cannot be 1, it follows that $n = p$ and thereby $p \mid (q-1)$.

5) Various compact representations for $V$ (including CRT-based representations) are discussed in [24, Appendix B].

6) The inversion-free CRT technique is demonstrated in [24, Sect. 3]. As presented, it further includes a randomness step to blind the input: $C' \leftarrow Cr \bmod N$ with $r \xleftarrow{\$} \mathbb{Z}_N^*$ and $C^d \bmod N = C'(py'_q + qy'_p) \bmod N$ where $y'_q \leftarrow \left(pr(C'p)^{e-1}\right)^{q-1-d_q} \bmod q$ and $y'_p \leftarrow \left(qr(C'q)^{e-1}\right)^{p-1-d_p} \bmod p$.

7) The algorithm for computing $D^{-1} \bmod 2^{2^n}$ is presented in [30, Fig. 1] (note however that there is a typo: the modulus should read $2^{2^i}$). Another algorithm can be found in [19, § 3.2]. As an application to Eq. (8.3) for an odd integer $e$, since $d \coloneqq e^{-1} \bmod f < 2^{2^F}$ where $F \coloneqq \lceil \log_2 \log_2 f \rceil$, $d$ can equivalently be obtained as $d \leftarrow M \cdot e^{-1} \bmod 2^{2^F}$ with $M = (1 + f(-f^{-1} \bmod e)) \bmod 2^{2^F}$.

8) Let $\hat{d} \coloneqq \frac{-(ed_p-1)(ed_q-1)+1}{e}$. It is easily verified that $e\hat{d} \equiv 1 \pmod{(p-1)}$ and $e\hat{d} \equiv 1 \pmod{(q-1)}$. Hence, $\hat{d} \bmod (p-1)(q-1)$ is a valid value for private exponent $d$ in standard mode.