CHAPTER 15

The "Naive" RSA Algorithm

Chapter Goals

- To understand the naive RSA encryption algorithm and the assumptions on which its security relies.
- To do the same for the naive RSA signature algorithm.
- To show why these naive versions cannot be considered secure.
- To explain Wiener's attack on RSA using continued fractions.
- To explain how to use Coppersmith's Theorem for finding small roots of modular polynomial equations to extend this attack to other situations.
- To introduce the notions of partial key exposure and fault analysis.

15.1. "Naive" RSA Encryption

The RSA algorithm was the world's first public key encryption algorithm, and it has stood the test of time remarkably well. The RSA algorithm is based on the difficulty of the RSA problem considered in Chapter 2, and hence it is based on the difficulty of finding the prime factors of large integers. However, we have seen that it may be possible to solve the RSA problem without factoring, hence the RSA algorithm is not based completely on the difficulty of factoring.

Suppose Alice wishes to enable anyone to send her secret messages, which only she can decrypt. She first picks two large secret prime numbers p and q. Alice then computes

$$N = p \cdot q.$$

Alice also chooses an encryption exponent e which satisfies

$$\gcd(e, (p-1) \cdot (q-1)) = 1.$$

It is common to choose $e = 3, 17$ or $65\,537$. Now Alice's public key is the pair $\mathfrak{pk} = (N, e)$, which she can publish in a public directory. To compute her private key Alice applies the extended Euclidean algorithm to e and $(p-1)(q-1)$ to obtain the decryption exponent d, which should satisfy

$$e \cdot d = 1 \pmod{(p-1)(q-1)}.$$

Alice keeps secret her private key, which is the triple $\mathfrak{sk} = (d, p, q)$. Actually, she could simply throw away p and q, and retain a copy of her public key which contains the integer N, but as we saw in Chapter 6 holding onto the prime factors can aid the exponentiation algorithm modulo N.

Now suppose Bob wishes to encrypt a message to Alice. He first looks up Alice's public key and represents the message as a number m which is strictly less than the public modulus N. The ciphertext is then produced by raising the message to the power of the public encryption exponent modulo the public modulus, i.e.

$$c \leftarrow m^e \pmod{N}.$$

N.P. Smart, *Cryptography Made Simple*, Information Security and Cryptography, DOI 10.1007/978-3-319-21936-3_15

On receiving c Alice can decrypt the ciphertext to recover the message by exponentiating by the private decryption exponent, i.e.

$$m \leftarrow c^d \pmod{N}.$$

This works since the group $(\mathbb{Z}/N\mathbb{Z})^*$ has order

$$\phi(N) = (p-1)(q-1)$$

and so, by Lagrange's Theorem,

$$x^{(p-1)\cdot(q-1)} = 1 \pmod{N},$$

for all $x \in (\mathbb{Z}/N\mathbb{Z})^*$. Thus, for some integer s we have

$$ed - s \cdot (p-1) \cdot (q-1) = 1,$$

and so

$$\begin{aligned} c^d &= (m^e)^d = m^{e\cdot d} \\ &= m^{1+s(p-1)\cdot(q-1)} = m \cdot m^{s\cdot(p-1)\cdot(q-1)} \\ &= m. \end{aligned}$$

To make things clearer let's consider a baby example. Choose $p = 7$ and $q = 11$, and so $N = 77$ and $(p-1)\cdot(q-1) = 6 \cdot 10 = 60$. We pick as the public encryption exponent $e = 37$, since we have $\gcd(37, 60) = 1$. Then, applying the extended Euclidean algorithm we obtain $d = 13$ since

$$37 \cdot 13 = 481 = 1 \pmod{60}.$$

Suppose the message we wish to transmit is given by $m = 2$, then to encrypt m we compute

$$c \leftarrow m^e \pmod{N} = 2^{37} \pmod{77} = 51,$$

whilst to decrypt the ciphertext c we compute

$$m \leftarrow c^d \pmod{N} = 51^{13} \pmod{77} = 2.$$

The security of RSA on first inspection relies on the difficulty of finding the private encryption exponent d given only the public key, namely the public modulus N and the public encryption exponent e. In Chapter 2 we showed that the RSA problem is no harder than FACTOR, hence if we can factor N then we can find p and q and hence we can calculate d. Hence, if factoring is easy we can break RSA. Currently it is recommended that one takes public moduli of size around 2048 bits to ensure medium-term security.

Recall, from Chapter 11, that for a public key algorithm the adversary always has access to the encryption algorithm, hence she can always mount a chosen plaintext attack. We can show that RSA encryption meets our weakest notion of security for public key encryption, namely OW-CPA, assuming that the RSA problem is hard. To show this we use the reduction arguments of previous chapters. This example is rather trivial but we labour the point since these arguments are used over and over again.

Lemma 15.1. *If the RSA problem is hard then the naive RSA encryption scheme is* OW-CPA *secure. In particular if A is an adversary which breaks the* OW-CPA *security of naive RSA encryption for RSA moduli of v bits in length, then there is an adversary B such that*

$$\mathrm{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathsf{RSA}(v)}(A) = \mathrm{Adv}_v\mathsf{RSA}(B).$$

Proof. We wish to give an algorithm which solves the RSA problem using an algorithm to break the RSA cryptosystem as an oracle. If we can show this then we can conclude that breaking the RSA cryptosystem is no harder than solving the RSA problem.

Recall that the RSA problem is given $N = p \cdot q$, e and $y \in (\mathbb{Z}/N\mathbb{Z})^*$, compute an x such that $x^e \pmod{N} = y$. We use our oracle to break the RSA encryption algorithm to "decrypt" the message

corresponding to $c = y$; this oracle will return the plaintext message m. Then our RSA problem is solved by setting $x \leftarrow m$ since, by definition,

$$m^e \pmod N = c = y.$$

So if we can break the RSA algorithm then we can solve the RSA problem. □

This is, however, the best we can hope for since naive RSA encryption is deterministic, and thus we have the following.

Lemma 15.2. *Naive RSA encryption is* not IND-CPA *secure.*

Proof. Recall from Chapter 11 that in the IND-CPA game the adversary produces two plaintexts, which we shall denote by m_0 and m_1. The challenger, then encrypts one of them to obtain the challenge ciphertext $c^* \leftarrow {m_b}^e \pmod N$, for some hidden bit $b \in \{0, 1\}$. The adversary's goal is then to determine b. This task can be easily accomplished since all the adversary needs to do is to compute $c \leftarrow m_1^e \pmod N$, and then

- if $c^* = c$ then the attacker knows that $m_b = m_1$,
- if $c^* \neq c$ then the attacker knows that $m_b = m_0$.

□

The problem is that the attacker has access to the encryption function; it is a public key scheme after all. But using a deterministic encryption function is not the only problem with RSA, since RSA is also malleable due to the homomorphic property.

Definition 15.3 (Homomorphic Property)**.** *An encryption scheme has the (multiplicative) homomorphic property if given the encryptions of m_1 and m_2 we can determine the encryption of $m_1 \cdot m_2$, without knowing m_1 or m_2.*

A similar definition can be given for additive homomorphisms as well. That RSA has the homomorphic property follows from the equation

$$(m_1 \cdot m_2)^e \pmod N = ((m_1{}^e \pmod N) \cdot (m_2{}^e \pmod N)) \pmod N.$$

One can use the homomorphic property to show that RSA is not even one-way secure under an adaptive chosen ciphertext attack.

Lemma 15.4. *Naive RSA encryption is* not OW-CCA *secure.*

Proof. Recall from Chapter 11 that the OW-CCA game is like the OW-CPA game, except that the adversary now has access to a decryption oracle which can decrypt any ciphertext, bar the target ciphertext c^*. Suppose the challenger gives the adversary the challenge ciphertext

$$c^* = (m^*)^e \pmod N.$$

The goal of the adversary is to find m^*. The adversary then creates the "related" ciphertext $c = 2^e \cdot c^*$ and asks her decryption oracle to decrypt c to produce m. Notice that this is a legal query under the rules of the game as $c \neq c^*$. The adversary can then compute

$$\begin{aligned} \frac{m}{2} &= \frac{c^d}{2} = \frac{(2^e \cdot c^*)^d}{2} \\ &= \frac{2^{e \cdot d} \cdot (c^*)^d}{2} = \frac{2 \cdot m^*}{2} = m^*. \end{aligned}$$

□

In Chapter 16 we will present variants of RSA encryption, as well as other public key encryption schemes, and show that they are secure in the sense of IND-CCA. It is these more advanced algorithms which one should use in practice, and this is why we have dubbed the above method of encrypting "naive" RSA.

15.1.1. Rabin Encryption: The obvious question is whether we can construct an OW-CPA secure scheme, which is efficient, and which is based on the factoring problem itself. The Rabin encryption scheme is one such scheme; it replaces the RSA map (which is a permutation on the group $(\mathbb{Z}/N\mathbb{Z})^*$) with a map which is not injective. The Rabin scheme, due to Michael Rabin, bases its security on the difficulty of extracting square roots modulo $N = p \cdot q$, for two large unknown primes p and q. Recall Theorem 2.4, which showed that this SQRROOT problem is polynomial-time equivalent to the factoring problem.

Despite these plus points the Rabin system is not used as much as the RSA system. It is, however, useful to study for a number of reasons, both historical and theoretical. The basic idea of the system is also used in some higher-level protocols.

Key Generation: We first choose prime numbers of the form $p = q = 3 \pmod 4$, since this makes extracting square roots modulo p and q very fast. The private key is then the pair $\mathfrak{sk} \leftarrow (p, q)$, and the public key is $\mathfrak{pk} \leftarrow N = p \cdot q$.

Encryption: To encrypt a message m using the above public key, in the Rabin encryption algorithm we compute $c \leftarrow m^2 \pmod N$. Hence, encryption involves one multiplication modulo N, and is therefore much faster than RSA encryption, even when one chooses a small RSA encryption exponent. Indeed, encryption in the Rabin encryption system is much faster than almost any other public key scheme.

Decryption: Decryption is far more complicated; essentially we want to compute the value of $m = \sqrt{c} \pmod N$. At first sight this uses no private information, but a moment's thought reveals that one needs the factorization of N to be able to find the square root. In particular to compute m one computes

$$m_p = \sqrt{c} \pmod p = c^{(p+1)/4} \pmod p = \pm 35,$$
$$m_q = \sqrt{c} \pmod q = c^{(q+1)/4} \pmod q = \pm 44,$$

and then combines m_p and m_q by the Chinese Remainder Theorem. There are however four possible square roots modulo N, since N is the product of two primes. Hence, on decryption we obtain *four* possible plaintexts. This means that we need to add redundancy to the plaintext before encryption in order to decide which of the four possible plaintexts corresponds to the intended one.

Example: Let the private key be given by $p = 127$ and $q = 131$, and the public key be given by $N = 16\,637$. To encrypt the message $m = 4\,410$ we compute

$$c = m^2 \pmod N = 16\,084.$$

To decrypt we evaluate the square root of c modulo p and q

$$m_p = \sqrt{c} \pmod p = \pm 35,$$
$$m_q = \sqrt{c} \pmod q = \pm 44.$$

Now we apply the Chinese Remainder Theorem to both $\pm 35 \pmod p$ and $\pm 44 \pmod q$ so as to find the square root of c modulo N,

$$s = \sqrt{c} \pmod N = \pm 4\,410 \text{ and } \pm 1\,616.$$

This leaves us with the four "messages"

$$1\,616,\ 4\,410,\ 12\,227,\ \text{ or } 15\,021.$$

It should be pretty clear that any adversary breaking the OW-CPA security of the Rabin encryption scheme can immediately be turned into an adversary to break the SQRROOT problem, and hence into an adversary to factor integers. So we have the following.

Theorem 15.5. *If A is an adversary against the* OW-CPA *security of the Rabin encryption scheme* Π *for moduli of size v bits, then there is an adversary B against the factoring problem with*

$$\mathrm{Adv}_{\Pi}^{\mathsf{OW\text{-}CPA}}(A) = 2 \cdot \mathrm{Adv}_{v}^{\mathsf{FACTOR}}(B),$$

the factor of two coming from Lemma 2.6.

It should also be pretty obvious that the scheme is not OW-CCA secure, since the scheme is obviously malleable in the same way that RSA was. In addition it is clearly not IND-CPA as it is deterministic, like RSA.

15.2. "Naive" RSA Signatures

The RSA encryption algorithm is particularly interesting since it can be used directly as a so-called signature algorithm with message recovery.

- The sender applies the RSA *decryption* transform to generate the signature, by taking the message and raising it to the private exponent d

$$s \leftarrow m^d \pmod{N}.$$

- The receiver then applies the RSA *encryption* transform to recover the original message

$$m \leftarrow s^e \pmod{N}.$$

But this raises the question; how do we check the validity of the signature? If the original message is in a natural language such as English then one can verify that the extracted message is also in the same natural language. But this is not a solution for all possible messages. Hence one needs to add redundancy to the message.

One, almost prehistoric, way of doing this in the early days of public key cryptography was the following. Suppose the message m is t bits long and the RSA modulus N is k bits long, with $t < k - 32$. We first pad m to the right with zeros to produce a string of length a multiple of eight. We then add $(k - t)/8$ bytes to the left of m to produce a byte-string

$$m \leftarrow 00\|01\|FF\|FF\ldots\|FF\|00\|m.$$

The signature is then computed via

$$m^d \pmod{N}.$$

When verifying the signature we ensure that the recovered value of m has the correct padding. This form of padding also seems to prevent the following trivial existential forgery attack. The attacker picks s at random and then computes

$$m \leftarrow s^e \pmod{N}.$$

The attacker then has the signature s on the message m.

Moreover, the padding scheme also seems to prevent selective forgeries, which are in some sense an even worse form of weakness. Without the padding scheme we can produce a selective forgery, using access to a signing oracle, as follows. Suppose the attacker wishes to produce a signature s on the message m. She first generates a random $m_1 \in (\mathbb{Z}/N\mathbb{Z})^*$ and computes

$$m_2 \leftarrow \frac{m}{m_1}.$$

Then the attacker asks her oracle to sign the messages m_1 and m_2. This results in two signatures s_1 and s_2 such that

$$s_i = {m_i}^d \pmod{N}.$$

The attacker can then compute the signature on the message m by computing

$$s \leftarrow s_1 \cdot s_2 \pmod{N}$$

since

$$\begin{aligned} s &= s_1 \cdot s_2 \pmod N \\ &= {m_1}^d \cdot {m_2}^d \pmod N \\ &= (m_1 \cdot m_2)^d \pmod N \\ &= m^d \pmod N. \end{aligned}$$

Here we have used once more the homomorphic property of the RSA function, just as we did when we showed that RSA encryption was not OW-CCA secure.

But not all messages will be so short so as to fit into the above method. Hence, naively to apply the RSA signature algorithm to a long message m we need to break it into blocks and sign each block in turn. This is very time-consuming for long messages. Worse than this, we must add serial numbers and more redundancy to each message otherwise an attacker could delete parts of the long message without us knowing, just as could happen when encrypting using a block cipher in ECB Mode. This problem arises because our signature model is one giving message recovery, i.e. the message is recovered from the signature and the verification process. If we used a model called a signature with *appendix* then we could first produce a hash of the message to be signed and then just sign the hash.

Using a cryptographic hash function H, such as those described in Chapter 14, it is possible to make RSA into a signature scheme without message recovery, which is very efficient for long messages. Suppose we are given a long message m for signing; we first compute $H(m)$ and then apply the RSA signing transform to $H(m)$, i.e. the signature is given by

$$s \leftarrow H(m)^d \pmod N.$$

The signature and message are then transmitted together as the pair (m, s). Verifying a message/signature pair (m, s) generated using a hash function involves three steps.

- "Encrypt" s using the RSA encryption function to recover h, i.e.

$$h \leftarrow s^e \pmod N.$$

- Compute $H(m)$ from m.
- Check whether $h = H(m)$. If they agree accept the signature as valid, otherwise the signature should be rejected.

Since a hash function does not usually have codomain the whole of the integers modulo N, in practice one needs to first hash and then pad the message. We could, for example, use the padding scheme given earlier when we discussed RSA with message recovery. If we assume the hash function produces a value in the range $[0, \ldots, N-1]$ then the above scheme is called RSA-FDH, for RSA *Full Domain Hash*. The name is because the codomain of the hash function is the entire domain of the RSA function.

Recall that when we discussed cryptographic hash functions we said that they should satisfy the following three properties:

(1) **Preimage Resistant:** It should be hard to find a message with a given hash value.
(2) **Collision Resistant:** It should be hard to find two messages with the same hash value.
(3) **Second Preimage Resistant:** Given one message it should be hard to find another message with the same hash value.

It turns out that all three properties are needed when using the RSA-FDH signing algorithm, as we shall now show.

Requirement for Preimage Resistance: The one-way property stops a cryptanalyst from cooking up a message with a given signature. For example, suppose we are using RSA-FDH but with a hash function which does not have the one-way property. We then have the following attack.

- The adversary computes
$$h \leftarrow r^e \pmod N$$
for some random integer r.
- The adversary also computes the preimage of h under H (recall we are assuming that H does not have the one-way property), i.e. Eve computes
$$m \leftarrow H^{-1}(h).$$

The adversary now has your signature (m, r) on the message m. Recall that such a forgery is called an existential forgery, since the attacker may not have any control over the contents of the message on which she has obtained a digital signature.

Requirement for Collision Resistance: This is needed to avoid the following attack, which is performed by the legitimate signer.

- The signer chooses two messages m and m' with $H(m) = H(m')$.
- They sign m and output the signature (m, s).
- Later they repudiate this signature, saying it was really a signature on the message m'.

As a concrete example one could have that m is an electronic cheque for 1 000 euros whilst m' is an electronic cheque for 10 euros.

Requirement for Second Preimage Resistance: This property is needed to stop the following attack.

- An attacker obtains your signature (m, s) on a message m.
- The attacker finds another message m' with $H(m') = H(m)$.
- The attacker now has your signature (m', s) on the message m'.

Thus, the security of any signature scheme which uses a cryptographic hash function will depend both on the security of the underlying hard mathematical problem, such as factoring or the discrete logarithm problem, and the security of the underlying hash function. In Chapter 16 we will present some variants of the RSA signature algorithm, as well as others, and show that they are secure in the sense of EUF-CMA.

15.3. The Security of RSA

In this section we examine in more detail the security of the RSA function, and the resulting "naive" encryption and signature algorithms. In particular we show how knowing the private key *is* equivalent to factoring (which should be contrasted with being able to invert the function, namely the RSA problem), how knowledge of $\phi(N)$ is also equivalent to factoring, how sharing a modulus can be a bad idea, and how also having a small public exponent could introduce problems as well.

15.3.1. Knowledge of the Private Exponent and Factoring: Whilst it is unclear whether breaking RSA, in the sense of inverting the RSA function, is equivalent to factoring, determining the private key d given the public information, N and e, is equivalent to factoring. The algorithm in the next proof is an example of a Las Vegas algorithm: It is probabilistic in nature in the sense that whilst it may not actually give an answer (or terminate), it is however guaranteed that when it does give an answer then that answer will always be correct.

Lemma 15.6. *If one knows the RSA decryption exponent d corresponding to the public key (N, e) then one can efficiently factor N.*

PROOF. Recall that for some integer s

$$e \cdot d - 1 = s \cdot (p-1) \cdot (q-1).$$

We first pick an integer $x \neq 0$, this is guaranteed to satisfy

$$x^{e \cdot d - 1} = 1 \pmod{N}.$$

We now compute a square root y_1 of one modulo N,

$$y_1 \leftarrow \sqrt{x^{e \cdot d - 1}} = x^{(e \cdot d - 1)/2},$$

which we can do since $e \cdot d - 1$ is known and will be even. We will then have the identity

$$y_1{}^2 - 1 = 0 \pmod{N},$$

which we can use to recover a factor of N via computing

$$\gcd(y_1 - 1, N).$$

But this will only work when $y_1 \neq \pm 1 \pmod{N}$.

Now suppose we are unlucky and we obtain $y_1 = \pm 1 \pmod{N}$ rather than a factor of N. If $y_1 = -1 \pmod{N}$, then we set $y_1 \leftarrow -y_1$. Thus we are always left with the case $y_1 = 1 \pmod{N}$. We take another square root of one via

$$y_2 \leftarrow \sqrt{y_1} = x^{(e \cdot d - 1)/4}.$$

Again we have

$$y_2{}^2 - 1 = y_1 - 1 = 0 \pmod{N}.$$

Hence we compute

$$\gcd(y_2 - 1, N)$$

and see whether this gives a factor of N. Again this will give a factor of N unless $y_2 = \pm 1$. If we are unlucky we repeat once more and so on.

This method can be repeated until either we have factored N or until $(e \cdot d - 1)/2^t$ is no longer divisible by 2. In this latter case we return to the beginning, choose a new random value of x and start again. □

We shall now present a small example of the previous method. Consider the following RSA parameters

$$N = 1\,441\,499, \quad e = 17 \text{ and } d = 507\,905.$$

Recall that we are assuming that the private exponent d is public knowledge. We will show that the previous method does in fact find a factor of N. Put

$$t_1 \leftarrow (e \cdot d - 1)/2 = 4\,317\,192,$$
$$x \leftarrow 2.$$

To compute y_1 we evaluate

$$y_1 \leftarrow x^{(e \cdot d - 1)/2} = 2^{t_1} = 1 \pmod{N}.$$

Since we obtain $y_1 = 1$ we need to set

$$t_2 \leftarrow t_1/2 = (e \cdot d - 1)/4 = 2\,158\,596,$$
$$y_2 \leftarrow 2^{t_2}.$$

We now compute y_2,

$$y_2 \leftarrow x^{(e \cdot d - 1)/4} = 2^{t_2} = 1 \pmod{N}.$$

So we need to repeat the method again; this time we obtain $t_3 = (e \cdot d - 1)/8 = 1\,079\,298$. We compute y_3,

$$y_3 = x^{(e \cdot d - 1)/8} = 2^{t_3} = 119\,533 \pmod{N}.$$

So

$$y_3{}^2 - 1 = (y_3 - 1) \cdot (y_3 + 1) = 0 \pmod{N},$$

and we compute a prime factor of N by evaluating $\gcd(y_3 - 1, N) = 1423$.

15.3.2. Knowledge of $\phi(N)$ and Factoring: We have seen that knowledge of d allows us to factor N. Now we will show that knowledge of $\Phi = \phi(N)$ also allows us to factor N.

Lemma 15.7. *Given an RSA modulus N and the value of $\Phi = \phi(N)$ one can efficiently factor N.*

Proof. We have

$$\Phi = (p - 1) \cdot (q - 1) = N - (p + q) + 1.$$

Hence, if we set $S = N + 1 - \Phi$, we obtain

$$S = p + q.$$

So we need to determine p and q from their sum S and product N. Define the polynomial

$$f(X) = (X - p) \cdot (X - q) = X^2 - S \cdot X + N.$$

So we can find p and q by solving $f(X) = 0$ using the standard formulae for extracting the roots of a quadratic polynomial,

$$p = \frac{S + \sqrt{S^2 - 4 \cdot N}}{2},$$
$$q = \frac{S - \sqrt{S^2 - 4 \cdot N}}{2}.$$

□

As an example consider the RSA public modulus $N = 18\,923$. Assume that we are given $\Phi = \phi(N) = 18\,648$. We then compute

$$S = p + q = N + 1 - \Phi = 276.$$

Using this we compute the polynomial

$$f(X) = X^2 - S \cdot X + N = X^2 - 276 \cdot X + 18\,923$$

and find that its roots over the real numbers are $p = 149$, $q = 127$, which are indeed the factors of N.

15.3.3. Use of a Shared Modulus: Since modular arithmetic is very expensive it can be very tempting to set up a system in which a number of users share the same public modulus N but use different public/private exponents, (e_i, d_i). One reason to do this could be to allow very fast hardware acceleration of modular arithmetic, specially tuned to the chosen shared modulus N. This is, however, a very silly idea since it can be attacked in one of two ways, either by a malicious insider or by an external attacker.

Suppose the attacker is one of the internal users, say user number one. She can now compute the value of the decryption exponent for user number two, namely d_2. First user one computes p and q since she knows d_1, via the algorithm in the proof of Lemma 15.6. Then user one computes $\phi(N) = (p - 1) \cdot (q - 1)$, and finally she can recover d_2 from

$$d_2 = \frac{1}{e_2} \pmod{\phi(N)}.$$

Now suppose the attacker is not one of the people who share the modulus, and that the two public exponents e_1 and e_2 are coprime. We now present an attack against the "naive" RSA encryption algorithm in this setting. Suppose Alice sends the same message m to two of the users

with public keys (N, e_1) and (N, e_2), i.e. $N_1 = N_2 = N$. Eve, the external attacker, sees the messages c_1 and c_2, where the ciphertexts are derived by executing

$$\begin{aligned} c_1 &\leftarrow m^{e_1} \pmod{N}, \\ c_2 &\leftarrow m^{e_2} \pmod{N}. \end{aligned}$$

Eve can now compute

$$\begin{aligned} t_1 &\leftarrow {e_1}^{-1} \pmod{e_2}, \\ t_2 &\leftarrow (t_1 \cdot e_1 - 1)/e_2, \end{aligned}$$

and can recover the message m from

$$\begin{aligned} c_1^{t_1} \cdot c_2^{-t_2} &= m^{e_1 \cdot t_1} m^{-e_2 \cdot t_2} \\ &= m^{1+e_2 \cdot t_2} m^{-e_2 \cdot t_2} \\ &= m^{1+e_2 \cdot t_2 - e_2 \cdot t_2} \\ &= m^1 = m. \end{aligned}$$

As an example of this external attack, take the public keys to be

$$N = N_1 = N_2 = 18\,923, \ e_1 = 11 \text{ and } e_2 = 5.$$

Now suppose Eve sees the ciphertexts

$$c_1 = 1514 \text{ and } c_2 = 8189$$

corresponding to the same plaintext m. Then Eve computes $t_1 = 1$ and $t_2 = 2$, and recovers the message

$$m = c_1^{t_1} \cdot c_2^{-t_2} = 100 \pmod{N}.$$

15.3.4. Use of a Small Public Exponent: In practice RSA systems often use a small public exponent e so as to cut down the computational cost of the sender. We shall now show that this can also lead to problems for the RSA encryption algorithm. Suppose we have three users all with different public moduli N_1, N_2 and N_3. In addition suppose they all have the same small public exponent $e = 3$. Suppose someone sends them the same message m. The attacker Eve sees the messages

$$\begin{aligned} c_1 &\leftarrow m^3 \pmod{N_1}, \\ c_2 &\leftarrow m^3 \pmod{N_2}, \\ c_3 &\leftarrow m^3 \pmod{N_3}. \end{aligned}$$

Now the attacker, using the Chinese Remainder Theorem, computes the simultaneous solution to the equations

$$X = c_i \pmod{N_i} \text{ for } i = 1, 2, 3,$$

to obtain

$$X = m^3 \pmod{N_1 \cdot N_2 \cdot N_3}.$$

But since $m^3 < N_1 \cdot N_2 \cdot N_3$ we must have $X = m^3$ identically over the integers. Hence we can recover m by taking the real cube root of X.

As a simple example of this attack; take $N_1 = 323$, $N_2 = 299$ and $N_3 = 341$. Suppose Eve sees the ciphertexts

$$c_1 = 50, \ c_2 = 268 \text{ and } c_3 = 1,$$

and wants to determine the common value of m. Eve computes via the Chinese Remainder Theorem

$$X = 300\,763 \pmod{N_1 \cdot N_2 \cdot N_3}.$$

Finally, she computes over the integers $m = X^{1/3} = 67$.

This attack and the previous one are interesting since we find the message without factoring the modulus. This is, albeit slight, evidence that breaking RSA is easier than factoring. The main lesson, however, from both these attacks is that plaintext should be randomly padded before transmission. That way the same "message" is never encrypted to two different people. In addition one should probably avoid very small exponents for encryption; $e = 65\,537$ is the usual choice now in use. However, small public exponents for RSA signatures produce no such problems. So for RSA signatures it is common to see $e = 3$ being used in practice.

15.4. Some Lattice-Based Attacks on RSA

In this section we examine how lattices can be used to attack certain systems, when some other side information is known.

15.4.1. Håstad's Attack: Earlier in this chapter we saw the following attack on the RSA system: Given three public keys (N_i, e_i) all with the same encryption exponent $e_i = 3$, if a user sent the same message to all three public keys then an adversary could recover the plaintext using the Chinese Remainder Theorem. Suppose that we try to protect against this attack by insisting that before encrypting a message m we first pad with some user-specific data. For example the ciphertext becomes, for user i,

$$c_i = (i \cdot 2^h + m)^3 \pmod{N_i}.$$

However, one can still break this system using an attack due to Håstad. Håstad's attack is related to Coppersmith's Theorem since we can interpret the attack scenario, generalized to k users and public encryption exponent e, as being given k polynomials of degree e

$$g_i(x) = (i \cdot 2^h + x)^e - c_i,\ 1 \le i \le k.$$

Then given that there is an m such that

$$g_i(m) = 0 \pmod{N_i},$$

the goal is to recover m. We can assume that m is smaller than any one of the moduli N_i. Setting

$$N = N_1 \cdot N_2 \cdots N_k$$

and using the Chinese Remainder Theorem we can find t_i so that

$$g(x) = \sum_{i=1}^{k} t_i \cdot g_i(x)$$

and

$$g(m) = 0 \pmod{N}.$$

Then, since g has degree e and is monic, using Theorem 5.10 we can recover m in polynomial time, as long as we have at least as many ciphertexts/users as the encryption exponent i.e. $k \ge e$, since

$$m < \min_i N_i < N^{1/k} \le N^{1/e}.$$

15.4.2. Franklin–Reiter Attack and Coppersmith's Generalization: Now suppose we have one RSA public key (N, e) owned by Alice. The Franklin–Reiter attack applies to the following situation: Bob wishes to send two related messages m_1 and m_2 to Alice, where the relation is given by the public polynomial

$$m_1 = f(m_2) \pmod{N}.$$

We shall see that, given c_1 and c_2, an attacker has a good chance of determining m_1 and m_2 for any small encryption exponent e. The attack is particularly simple when

$$f = a \cdot x + b \text{ and } e = 3,$$

with a and b fixed and given to the attacker. The attacker knows that m_2 is a root, modulo N, of the two polynomials

$$g_1(x) = x^3 - c_2,$$
$$g_2(x) = f(x)^3 - c_1.$$

So the linear factor $x - m_2$ divides both $g_1(x)$ and $g_2(x)$.

We now form the greatest common divisor of $g_1(x)$ and $g_2(x)$. Strictly speaking this is not possible in general since $(\mathbb{Z}/N)\mathbb{Z}[x]$ is not a Euclidean ring, but if the Euclidean algorithm breaks down then we would find a factor of N and so be able to find Alice's private key in any case. One can show that when $f = a \cdot x + b$ and $e = 3$, the resulting gcd, when it exists, must always be the linear factor $x - m_2$, and so the attacker can always find m_2 and then m_1.

Coppersmith extended the attack of Franklin and Reiter in a way which also extends the padding result from Håstad's attack. Suppose before sending a message m we pad it with some random data. So for example if N is an n-bit RSA modulus and m is a k-bit message then we could append $n - k$ random bits to either the top or bottom of the message. Say

$$m' = 2^{n-k} \cdot m + r$$

where r is some, per message, random number of length $n-k$. This would seem to be a good idea in any case, since it makes the RSA function randomized, which might help in making it semantically secure. However, Coppersmith showed that this naive padding method is insecure.

Suppose Bob sends the same message to Alice twice, i.e. we have ciphertexts c_1 and c_2 corresponding to the messages

$$m_1 = 2^{n-k} \cdot m + r_1,$$
$$m_2 = 2^{n-k} \cdot m + r_2,$$

where r_1, r_2 are two different random $(n-k)$-bit numbers. The attacker sets $y_0 = r_2 - r_1$ and is led to solve the simultaneous equations

$$g_1(x, y) = x^e - c_1,$$
$$g_2(x, y) = (x + y)^e - c_2.$$

The attacker forms the resultant $h(y)$ of $g_1(x, y)$ and $g_2(x, y)$ with respect to x. Now $y_0 = r_2 - r_1$ is a small root of the polynomial $h(y)$, which has degree e^2. Using Coppersmith's Theorem 5.10 the attacker recovers $r_2 - r_1$ and then recovers m_2 using the method of the Franklin–Reiter attack.

Whilst the above trivial padding scheme is therefore insecure, one can find secure padding schemes for the RSA encryption algorithm. We shall return to padding schemes for RSA in Chapter 16.

15.4.3. Wiener's Attack on RSA: We have mentioned that often one uses a small public RSA exponent e so as to speed up the public key operations in RSA. Sometimes we have applications where it is more important to have a fast private key operation. Hence, one could be tempted to choose a small value of the private exponent d. Clearly this will lead to a large value of the encryption exponent e and we cannot choose too small a value for d, otherwise an attacker could find d using exhaustive search. However, it turns out that d needs to be at least the size of $\frac{1}{3} \cdot N^{1/4}$ due to an ingenious attack by Wiener which uses continued fractions.

Wiener's attack uses continued fractions as follows. We assume we have an RSA modulus $N = p \cdot q$ with $q < p < 2q$. In addition assume that the attacker knows that we have a small

decryption exponent $d < \frac{1}{3} \cdot N^{1/4}$. The encryption exponent e is given to the attacker, where this exponent satisfies

$$e \cdot d = 1 \pmod{\Phi},$$

with

$$\Phi = \phi(N) = (p-1) \cdot (q-1).$$

We also assume $e < \Phi$, since this holds in most systems. First notice that this means there is an integer k such that

$$e \cdot d - k \cdot \Phi = 1.$$

Hence, we have

$$\left| \frac{e}{\Phi} - \frac{k}{d} \right| = \frac{1}{d \cdot \Phi}.$$

Now, $\Phi \approx N$, since

$$|N - \Phi| = |p + q - 1| < 3 \cdot \sqrt{N}.$$

So we should have that $\frac{e}{N}$ is a close approximation to $\frac{k}{d}$.

$$\begin{aligned}
\left| \frac{e}{N} - \frac{k}{d} \right| &= \left| \frac{e \cdot d - N \cdot k}{d \cdot N} \right| \\
&= \left| \frac{e \cdot d - k \cdot \Phi - N \cdot k + k \cdot \Phi}{d \cdot N} \right| \\
&= \left| \frac{1 - k \cdot (N - \Phi)}{d \cdot N} \right| \\
&\leq \left| \frac{3 \cdot k \cdot \sqrt{N}}{d \cdot N} \right| \\
&= \frac{3 \cdot k}{d \cdot \sqrt{N}}.
\end{aligned}$$

Since $e < \Phi$, it is clear that we have $k < d$, which is itself less than $\frac{1}{3} \cdot N^{1/4}$ by assumption. Hence,

$$\left| \frac{e}{N} - \frac{k}{d} \right| < \frac{1}{2 \cdot d^2}.$$

Since $\gcd(k, d) = 1$ we see that $\frac{k}{d}$ will be a fraction in its lowest terms. Hence, the fraction

$$\frac{k}{d}$$

must arise as one of the convergents of the continued fraction expansion of

$$\frac{e}{N}.$$

The correct one can be detected by simply testing which one gives a d which satisfies

$$(m^e)^d = m \pmod{N}$$

for some random value of m. The total number of convergents we will need to take is of order $O(\log N)$, hence the above gives a linear-time algorithm to determine the private exponent when it is less than $\frac{1}{3} \cdot N^{1/4}$.

As an example suppose we have the RSA modulus

$$N = 9\,449\,868\,410\,449$$

with the public key

$$e = 6\,792\,605\,526\,025.$$

We are told that the decryption exponent satisfies $d < \frac{1}{3} \cdot N^{1/4} \approx 584$. To apply Wiener's attack we compute the continued fraction expansion of the number

$$\alpha = \frac{e}{N},$$

and check each denominator of a convergent to see whether it is equal to the private key d. The convergents of the continued fraction expansion of α are given by

$$1, \frac{2}{3}, \frac{3}{4}, \frac{5}{7}, \frac{18}{25}, \frac{23}{32}, \frac{409}{569}, \frac{1659}{2308}, \cdots$$

Checking each denominator in turn we see that the decryption exponent is given by

$$d = 569,$$

which is the denominator of the seventh convergent.

15.4.4. Extension to Wiener's Attack: Boneh and Durfee, using an analogue of the bivariate case of Coppersmith's Theorem 5.10, extended Wiener's attack to the case where

$$d \leq N^{0.292},$$

using a heuristic algorithm, i.e. the range of "bad" values of d was extended further. We do not go into the details but show how Boneh and Durfee proceed to a problem known as the small inverse problem. Suppose we have an RSA modulus N, with encryption exponent e and decryption exponent d. By definition there is an integer k such that

$$e \cdot d + \frac{k \cdot \Phi}{2} = 1,$$

where $\Phi = \phi(N)$. Expanding the definition of Φ we find

$$e \cdot d + k \cdot \left(\frac{N+1}{2} - \frac{p+q}{2} \right) = 1.$$

We set

$$s = -\frac{p+q}{2},$$
$$A = \frac{N+1}{2}.$$

Then finding d, where d is small, say $d < N^{\delta}$, is equivalent to finding the two small solutions k and s to the following congruence

$$f(k, s) = k \cdot (A + s) = 1 \pmod{e}.$$

To see that k and s are small relative to the modulus e for the above equation, notice that $e \approx N$ since d is small, and so

$$|s| < 2 \cdot N^{0.5} \approx e^{0.5} \text{ and } |k| < \frac{2 \cdot d \cdot e}{\Phi} \leq \frac{3 \cdot d \cdot e}{N} \approx e^{\delta}.$$

We can interpret this problem as finding an integer which is close to A whose inverse is small modulo e. This is called the small inverse problem. Boneh and Durfee show that this problem has a solution when $\delta \leq 0.292$, hence extending Wiener's attack. This is done by applying the multivariate analogue of Coppersmith's method to the polynomial $f(k, s)$.

15.5. Partial Key Exposure Attacks on RSA

Partial key exposure is related to the following question: Suppose in some cryptographic scheme the attacker recovers a certain set of bits of the private key, can the attacker use this to recover the whole private key? In other words, does partial exposure of the key result in a total break of the system? We shall present a number of RSA examples, however these are not the only ones. There are a number of results related to partial key exposure which relate to other schemes such as DSA or symmetric-key-based systems.

15.5.1. Partial Exposure of the MSBs of the RSA Decryption Exponent: Somewhat surprisingly for RSA, in the more common case of using a small public exponent e, one can trivially recover half of the bits of the private key d, namely the most significant ones, as follows. Recall that there is a value of k such that $0 < k < e$ with

$$e \cdot d - k \cdot (N - (p + q) + 1) = 1.$$

Now suppose for each possible value of i, $0 < i \leq e$, the attacker computes

$$d_i = \lfloor (i \cdot N + 1)/e \rfloor.$$

Then we have

$$|d_k - d| \leq k \cdot (p + q)/e \leq 3 \cdot k \cdot \sqrt{N}/e < 3 \cdot \sqrt{N}.$$

Hence, d_k is a good approximation for the actual value of d.

Now when $e = 3$ it is clear that with high probability we have $k = 2$ and so d_2 reveals half of the most significant bits of d. Unluckily for the attack, and luckily for the user, there is no known way to recover the rest of d given only the most significant bits.

15.5.2. Partial Exposure of Some Bits of the RSA Prime Factors: Suppose our n-bit RSA modulus N is given by $p \cdot q$, with $p \approx q$, and that the attacker has found the $n/4$ least significant bits of p. Recall that p is only around $n/2$ bits long in any case, so this means the attacker is given the lower half of all the bits making up p. We write

$$p = x_0 \cdot 2^{n/4} + p_0.$$

We then have, writing $q = y_0 \cdot 2^{n/4} + q_0$,

$$N = p_0 \cdot q_0 \pmod{2^{n/4}}.$$

Hence, we can determine the value of q_0. We now write down the polynomial

$$\begin{aligned} p(x, y) &= (p_0 + 2^{n/4} \cdot x) \cdot (q_0 + 2^{n/4} \cdot y) \\ &= p_0 \cdot q_0 + 2^{n/4} \cdot (p_0 \cdot y + q_0 \cdot x) + 2^{n/2} \cdot x \cdot y. \end{aligned}$$

Now $p(x, y)$ is a bivariate polynomial of degree two which has known small solution modulo N, namely (x_0, y_0) where $0 < x_0, y_0 \leq 2^{n/4} \approx N^{1/4}$. Hence, using the heuristic bivariate extension of Coppersmith's Theorem 5.10, we can recover x_0 and y_0 in polynomial time and so factor the modulus N. A similar attack applies when the attacker knows the $n/4$ most significant bits of p.

15.5.3. Partial Exposure of the LSBs of the RSA Decryption Exponent: We now suppose we are given, for small public exponent e, a quarter of the least significant bits of the private exponent d. That is we have d_0 where

$$d = d_0 + 2^{n/4} \cdot x_0$$

where $0 \leq x_0 \leq 2^{3 \cdot n/4}$. Recall that there is a value of k with $0 < k < e$ such that

$$e \cdot d - k \cdot (N - (p + q) + 1) = 1.$$

We then have, since $N = p \cdot q$,

$$e \cdot d \cdot p - k \cdot p \cdot (N - p + 1) + k \cdot N = p.$$

If we set $p_0 = p \pmod{2^{n/4}}$ then we have the equation

$$e \cdot d_0 \cdot p_0 - k \cdot p_0 \cdot (N - p_0 + 1) + k \cdot N - p_0 = 0 \pmod{2^{n/4}}. \tag{22}$$

This gives us the following algorithm to recover the whole of d. For each value of k less than e we solve equation (22) modulo $2^{n/4}$ for p_0. Each value of k will result in $O(\frac{n}{4})$ possible values for p_0. Using each of these values of p_0 in turn, we apply the previous technique for factoring N from Section 15.5.2. One such value of p_0 will be the correct value of $p \pmod{2^{n/4}}$ and so the above factorization algorithm will work and we can recover the value of d.

15.6. Fault Analysis

An interesting class of attacks results from trying to induce faults within a cryptographic system. We shall describe this area in relation to the RSA signature algorithm but similar attacks can be mounted on other cryptographic algorithms, both public and symmetric key. Imagine we have a hardware implementation of RSA, in a smart card say. On input of some message m the chip will sign the message for us, using some internal RSA private key. The attacker wishes to determine the private key hidden within the smart card. To do this the attacker can try to make the card perform some of the calculation incorrectly, by either altering the card's environment by heating or cooling it or by damaging the circuitry of the card in some way.

An interesting case is when the card uses the Chinese Remainder Theorem to perform the signing operation, to increase efficiency, as explained in Chapter 6. The card first computes the hash of the message

$$h = H(m).$$

Then the card computes

$$s_p = h^{d_p} \pmod{p},$$
$$s_q = h^{d_q} \pmod{q},$$

where $d_p = d \pmod{p-1}$ and $d_q = d \pmod{q-1}$. The final signature is produced by the card from s_p and s_q via the Chinese Remainder Theorem using

- $u = (s_q - s_p) \cdot T \pmod{q}$,
- $s = s_p + u \cdot p$,

where $T = p^{-1} \pmod{q}$.

Now suppose that the attacker can introduce a fault into the computation so that s_p is computed incorrectly. The attacker will then obtain a value of s such that

$$s^e \neq h \pmod{p},$$
$$s^e = h \pmod{q}.$$

Hence, by computing

$$q = \gcd(s^e - h, N)$$

she can factor the modulus.

Chapter Summary

- RSA is the most popular public key encryption algorithm, but its security rests on the difficulty of the RSA problem and not quite on the difficulty of FACTOR.

- Rabin encryption is based on the difficulty of extracting square roots modulo a composite modulus. Since the problems SQRROOT and FACTOR are polynomial-time equivalent this means that Rabin encryption is based on the difficulty of FACTOR.
- Naive RSA is not IND-CPA secure.
- The RSA encryption algorithm can be used in reverse to produce a public key signature scheme, but one needs to combine the RSA algorithm with a hash algorithm to obtain security for both short and long messages.
- Using a small private decryption exponent in RSA is not a good idea due to Wiener's attack.
- The Håstad and Franklin–Reiter attacks imply that one needs to be careful when designing padding schemes for RSA.
- Using Coppersmith's Theorem one can show that revealing a proportion of the bits of either p, q or d in RSA can lead to a complete break of the system.

Further Reading

Still the best quick introduction to the concept of public key cryptography can be found in the original paper of Diffie and Hellman; see also the original paper on RSA encryption. Our treatment of attacks on RSA in this chapter has closely followed the survey article by Boneh.

D. Boneh. *Twenty years of attacks on the RSA cryptosystem.* Notices of the American Mathematical Society (AMS), **46**, 203–213, 1999.

W. Diffie and M. Hellman. *New directions in cryptography.* IEEE Trans. on Info. Theory, **22**, 644–654, 1976.

R.L. Rivest, A. Shamir and L.M. Adleman. *A method for obtaining digital signatures and public-key cryptosystems.* Comm. ACM, **21**, 120–126, 1978.