Chapter 24

The RSA and Rabin Cryptosystems

The aim of this chapter is to briefly present some cryptosystems whose security is based on computational assumptions related to the integer factorisation problem. In particular, we study the RSA and Rabin cryptosystems. We also present some security arguments and techniques for efficient implementation.

Throughout the chapter we take 3072 bits as the benchmark length for an RSA modulus. We make the assumption that the cost of factoring a 3072-bit RSA modulus is 2^{128} bit operations. These figures should be used as a very rough guideline only.

24.1 The Textbook RSA Cryptosystem

Figure 24.1 recalls the "textbook" RSA cryptosystem, which was already presented in Section 1.2. We remind the reader that the main application of RSA encryption is to transport symmetric keys, rather than to encrypt actual documents. For digital signatures we always sign a hash of the message, and it is necessary that the hash function used in signatures is collision-resistant.

In Section 1.3 we noted that the security parameter κ is not necessarily the same as the bit-length of the RSA modulus. In this chapter it will be convenient to ignore this, and use the symbol κ to denote the bit-length of an RSA modulus N. We always assume that κ is even.

As we have seen in Section 1.2, certain security properties can only be satisfied if the encryption process is randomised. Since the RSA encryption algorithm is deterministic it follows that the message m used in RSA encryption should be obtained from some **randomised padding scheme**. For example, if N is a 3072-bit modulus then the "message" itself may be a 256-bit AES key and may have 2815 random bits appended to

KeyGen(κ): (Assume κ even.) Generate two distinct primes p and q uniformly at random in the range $2^{\kappa/2-1} < p, q < 2^{\kappa/2}$. Set $N = pq$ so that $2^{\kappa-2} < N < 2^{\kappa}$ is represented by κ bits (see Exercise 24.1.1 to ensure that N has leading bit 1).
Choose a random κ-bit integer e coprime to $(p-1)$ and $(q-1)$ (or choose $e = 2^{16}+1 = 65537$ and insist $p, q \not\equiv 1 \pmod{e}$). Define $d = e^{-1} \pmod{\lambda(N)}$ where $\lambda(N) = \mathrm{lcm}(p-1, q-1)$ is the **Carmichael lambda function**. Output the public key $\mathsf{pk} = (N, e)$ and the private key $\mathsf{sk} = (N, d)$.
Renaming p and q, if necessary, we may assume that $p < q$. Then $p < q < 2p$ and so $\sqrt{N/2} < p < \sqrt{N}$.

In textbooks, the message space and ciphertext space are usually taken to be $\mathsf{C}_\kappa = \mathsf{M}_\kappa = (\mathbb{Z}/N\mathbb{Z})^*$, but it fits Definition 1.3.1 better (and is good training) to define them to be $\mathsf{C}_\kappa = \{0,1\}^\kappa$ and $\mathsf{M}_\kappa = \{0,1\}^{\kappa-2}$ or $\{0,1\}^{\kappa-1}$.

Encrypt($\mathsf{m}, (N, e)$): Assume that $\mathsf{m} \in \mathsf{M}_\kappa$.

- Compute $\mathsf{c} = \mathsf{m}^e \pmod{N}$ (see later for padding schemes).
- Return the ciphertext c.

Decrypt($\mathsf{c}, (N, d)$): Compute $\mathsf{m} = \mathsf{c}^d \pmod{N}$ and output either m, or $\perp$ if $\mathsf{m} \notin \mathsf{M}_\kappa$.

Sign($\mathsf{m}, (N, d)$): Compute $\mathsf{s} = \mathsf{m}^d \pmod{N}$.

Verify($\mathsf{m}, \mathsf{s}, (N, e)$): Check whether $\mathsf{m} \equiv \mathsf{s}^e \pmod{N}$.

Figure 24.1: Textbook RSA Public Key Encryption and Signature Schemes.

it. More elaborate padding schemes will be described in Section 24.7.2.

Exercise 24.1.1. Give a KeyGen algorithm that takes as input a security parameter κ (assumed to be even) and an l-bit string u (where $l < \kappa/2$) and outputs a κ-bit product $N = pq$ of two $\kappa/2$-bit primes such that the l most significant bits of N are equal to u. In particular, one can ensure that $2^{\kappa-1} < N < 2^{\kappa}$ and so N is a κ-bit integer.

Exercise 24.1.2. Let $N \in \mathbb{N}$. Prove that the Carmichael function $\lambda(N)$ divides the Euler function $\varphi(N)$. Prove that RSA decryption does return the message.

An odd prime p is a **safe prime** (or **Sophie-Germain prime**) if $(p-1)/2$ is also prime (see Exercise 12.2.10). For certain applications it is necessary to restrict to RSA moduli that are products of safe primes (usually so that $(\mathbb{Z}/N\mathbb{Z})^*$ has no elements of small order, except for order 2). It is conjectured that there is some constant $c > 0$ such that, for all sufficiently large $k \in \mathbb{N}$, there are at least $c2^k/k^2$ safe primes p such that $2^{k-1} < p < 2^k$.

Exercise 24.1.3. Once upon a time it was thought to be necessary to insist that p and q be strong primes for RSA.[1] A prime p is a **strong prime** if $p-1$ and $p+1$ have large prime factors r_1 and r_2 respectively, and $r_1 - 1$ has a large prime factor r_3. It is conjectured that there are infinitely many strong primes. Give an algorithm that takes as input integers k, k_1, k_2, k_3 such that $k_3 < k_1 < k$ and $k_2 < k$ and generates a k-bit strong prime p such that each prime r_i as above has k_i bits.

[1] This was so that $N = pq$ would not succumb to certain special purpose factoring algorithms. Nowadays it is realised that if p and q are chosen uniformly at random then no special purpose factoring algorithm will be successful with noticeable probability and so it is unnecessary to test for special cases.

24.1.1 Efficient Implementation of RSA

As we have seen in Section 12.2, $\kappa/2$-bit probable primes can be found in expected time of $O(\kappa^5)$ bit operations (or $O(\kappa^{4+o(1)})$ using fast arithmetic). One can make this provable using the AKS method, with asymptotic complexity $O(\kappa^{5+o(1)})$ bit operations using fast arithmetic. In any case, RSA key generation is polynomial-time. A more serious challenge is to ensure that encryption and decryption (equivalently, signing and verification) are as fast as possible.

Encryption and decryption are exponentiation modulo N and thus require $O(\log(N)M(\log(N)))$ bit operations, which is polynomial-time. For current RSA key sizes, Karatsuba multiplication is most appropriate, hence one should assume that $M(\log(N)) = \log(N)^{1.58}$. Many of the techniques mentioned in earlier chapters to speed up exponentiation can be used in RSA, particularly sliding window methods. Since e and d are fixed one can also pre-compute addition chains to minimise the cost of exponentiation.

In practice the following two improvements are almost always used.

- **Small public exponents** e (also called **low-exponent RSA**). Traditionally $e = 3$ was proposed, but these days $e = 65537 = 2^{16} + 1$ is most common. Encryption requires only 16 modular squarings and a modular multiplication.
- Use the Chinese remainder theorem (CRT) to decrypt.[2] Let $d_p \equiv e^{-1} \pmod{p-1}$ and $d_q \equiv e^{-1} \pmod{q-1}$. These are called the **CRT private exponents**. Given a ciphertext c one computes $\mathsf{m}_p = \mathsf{c}^{d_p} \pmod{p}$ and $\mathsf{m}_q = \mathsf{c}^{d_q} \pmod{q}$. The message m is then computed using the Chinese remainder theorem (it is convenient to use the method of Exercise 2.6.3 for the CRT).

  For this system the private key is then $\mathsf{sk} = (p, q, d_p, d_q)$. If we denote by $T = c\log(N)M(\log(N))$ the cost of a single exponentiation modulo N to a power $d \approx N$ then the cost using the Chinese remainder theorem is approximately $2c(\log(N)/2)M(\log(N)/2)$ (this is assuming the cost of the Chinese remaindering is negligible). When using Karatsuba multiplication this speeds up RSA decryption by a factor of approximately 3 (in other words, the new running time is a third of the old running time).

24.1.2 Variants of RSA

There has been significant effort devoted to finding more efficient variants of the RSA cryptosystem. We briefly mention some of these now.

Example 24.1.4. (**Multiprime-RSA**[3]) Let $p_1, \ldots, p_k$ be primes of approximately κ/k bits and let $N = p_1 \cdots p_k$. One can use N as a public modulus for the RSA cryptosystem. Using the Chinese remainder theorem for decryption has cost roughly the same as k exponentiations to powers of bit-length κ/k and modulo primes of bit-length κ/k. Hence, the speedup is roughly by a factor $k/k^{2.58} = 1/k^{1.58}$.

To put this in context, going from a single exponentiation to using the Chinese remainder theorem in the case of 2 primes gave a speedup by a factor of 3. Using 3 primes gives an overall speedup by a factor of roughly 5.7, which is a further speedup of a factor 1.9 over the 2-prime case. Using 4 primes gives an overall speedup of about 8.9, which is an additional speedup over 3 primes by a factor 1.6.

However, there is a limit to how large k can be, as the complexity of the elliptic curve factoring method mainly depends on the size of the smallest factor of N.

[2] This idea is often credited to Quisquater and Couvreur [493] but it also appears in Rabin [494].
[3] This idea was proposed (and patented) by Collins, Hopkins, Langford and Sabin.

Exercise 24.1.5. (Tunable balancing of RSA) An alternative approach is to construct the public key $(N = pq, e)$ so that the Chinese remainder decryption exponents are relatively short. The security of this system will be discussed in Section 24.5.2.

Let κ, n_e, n_d be the desired bit-lengths of N, e and $d \pmod{p-1}, d \pmod{q-1}$. Assume that $n_e + n_d > \kappa/2$. Give an algorithm to generate primes p and q of bit-length $\kappa/2$, integers d_p and d_q of bit-length n_d and an integer e of bit-length n_e such that $ed_p \equiv 1 \pmod{p-1}$ and $ed_q \equiv 1 \pmod{q-1}$.

The fastest variant of RSA is due to Takagi and uses moduli of the form $N = p^r q$. For some discussion about factoring such integers see Section 19.4.3.

Example 24.1.6. (**Takagi-RSA** [599]) Let $N = p^r q$ where p and q are primes and $r > 1$. Suppose the public exponent e in RSA is small. One can compute $\mathsf{c}^d \pmod{N}$ as follows. Let $d_p \equiv d \pmod{p-1}$ and $d_q \equiv d \pmod{q-1}$. One first computes $\mathsf{m}_p = \mathsf{c}^{d_p} \pmod{p}$ and $\mathsf{m}_q = \mathsf{c}^{d_q} \pmod{q}$.

To determine $\mathsf{m} \pmod{p^r}$ one uses Hensel lifting. If we have determined $\mathsf{m}_i = \mathsf{m} \pmod{p^i}$ such that $\mathsf{m}_i^e \equiv \mathsf{c} \pmod{p^i}$ then we lift to a solution modulo p^{i+1} by writing $\mathsf{m}_{i+1} = \mathsf{m}_i + xp^i$, where x is a variable. Then

$$\mathsf{m}_{i+1}^e = (\mathsf{m}_i + xp^i)^e \equiv \mathsf{m}_i^e + x(e\mathsf{m}_i^{e-1})p^i \equiv \mathsf{c} \pmod{p^{i+1}} \tag{24.1}$$

gives a linear equation in x modulo p. Note that computing $\mathsf{m}_i^e \pmod{p^{i+1}}$ in equation (24.1) is only efficient when e is small. If e is large then the Hensel lifting stage is no faster than just computing $\mathsf{c}^{e^{-1} \pmod{\varphi(p^r)}} \pmod{p^r}$ directly.

The total cost is two "full" exponentiations to compute m_p and m_q, $r-1$ executions of the Hensel lifting stage, plus one execution of the Chinese remainder theorem. Ignoring everything except the two big exponentiations one has an algorithm whose cost is $2/(r+1)^{2.58}$ times faster than naive textbook RSA decryption. Taking $r = 2$ this is about 9 times faster than standard RSA (i.e., about 1.6 times faster than using 3-prime RSA) and taking $r = 3$ is about 18 times faster than standard RSA (i.e., about 2 times faster than using 4-prime RSA).

Exercise 24.1.7. Let $N = (2^{20} + 7)^3(2^{19} + 21)$ and let $\mathsf{c} = 474776119073176490663504$ be the RSA encryption of a message m using public exponent $e = 3$. Determine the message using the Takagi decryption algorithm.

Exercise 24.1.8. Describe and analyse the RSA cryptosystem using moduli of the form $N = p^r q^s$. Explain why it is necessary that $r \neq s$.

Exercise 24.1.9. Write pseudocode for Takagi-RSA decryption.

Exercise 24.1.10. (Shamir's RSA for paranoids [548]) Let $N = pq$ where q is much larger than p (for example, $p \approx 2^{500}$ and $q \approx 2^{4500}$). The assumption is that factoring numbers of this form is much harder than factoring $N = pq$ where $p, q \approx 2^{500}$. Suppose one is encrypting a (padded) message m such that $1 \leq \mathsf{m} < p$ and suppose we use public exponent $e > 2\log_2(N)/\log_2(p)$ (so that, typically, $\mathsf{m}^e \approx N^2$). Encryption is computing $\mathsf{c} = \mathsf{m}^e \pmod{N}$ as usual. Shamir's observation is that one can decrypt by computing $\mathsf{m} = \mathsf{c}^{d_p} \pmod{p}$ where $ed_p \equiv 1 \pmod{p-1}$.

How much faster is this than RSA decryption using CRT with a 5000-bit modulus if the primes have equal size? If no padding scheme is used (i.e., every $1 \leq \mathsf{m} < p$ is a valid message) give an adaptive (CCA1) attack on this scheme that yields the factorisation of a user's modulus.

24.1.3 Security of Textbook RSA

We have presented "textbook" RSA above. This is unsuitable for practical applications for many reasons. In practice, RSA should only be used with a secure randomised padding scheme. Nevertheless, it is instructive to consider the security of textbook RSA with respect to the security definitions presented earlier.

Exercise 1.3.4 showed that textbook RSA encryption does not have OWE-CCA security and Exercise 1.3.9 showed that textbook RSA signatures do not have existential forgery security even under a passive attack. We recall one more easy attack.

Exercise 24.1.11. Show that one can use the Jacobi symbol to attack the IND-CPA security of RSA encryption.

Despite the fact that RSA is supposed to be related to factoring, the security actually relies on the following computational problem.

Definition 24.1.12. Let N, e be such that $\gcd(e, \lambda(N)) = 1$. The **RSA problem** (also called the **e-th roots problem**) is: Given $y \in (\mathbb{Z}/N\mathbb{Z})^*$ to compute x such that $x^e \equiv y \pmod{N}$.

It is clear that the RSA problem is not harder than factoring.

Lemma 24.1.13. *The OWE-CPA security of textbook RSA is equivalent to the RSA problem.*

Proof: (Sketch) We show that an algorithm to break OWE-CPA security of textbook RSA can be used to build an algorithm to solve the RSA problem. Let A be an adversary against the OWE-CPA security of RSA. Let (N, e, c) be a challenge RSA problem instance. If $1 < \gcd(\mathsf{c}, N) < N$ then split N and solve the RSA problem. Otherwise, call the adversary A on the public key (N, e) and offer the challenge ciphertext c. If A returns the message m then we are done. If A returns $\perp$ (e.g., because the decryption of c does not lie in M_κ) then replace c by $\mathsf{c}r^e \pmod{N}$ for a random $1 < r < N$ and repeat. When $\mathsf{M}_\kappa = \{0,1\}^{\kappa-2}$ then, with probability at least $1/4$, the reduction will succeed, and so one expects to perform 4 trials. The converse is also immediate. □

Exercise 24.1.14. Show that textbook RSA has selective signature forgery under passive attacks if and only if the RSA problem is hard.

One of the major unsolved problems in cryptography is to determine the relationship between the RSA problem and factoring. There is no known reduction of factoring to breaking RSA. Indeed, there is some indirect evidence that breaking RSA with small public exponent e is not as hard as factoring: Boneh and Venkatesan [87] show that an efficient "algebraic reduction"[4] from FACTOR to low-exponent RSA can be converted into an efficient algorithm for factoring. Similarly, Coppersmith [141] shows that some variants of the RSA problem, where e is small and only a small part of an e-th root x is unknown, are easy (see Exercise 19.1.15).

Definition 24.1.15 describes some computational problems underlying the security of RSA. The reader is warned that some of these names are non-standard.

Definition 24.1.15. Let S be a set of integers, for example $S = \mathbb{N}$ or $S = \{pq : p \text{ and } q \text{ are primes such that } p < q < 2p\}$. We call the latter set the **set of RSA moduli**.

[4] We do not give a formal definition. Essentially this is an algorithm that takes as input N, queries an oracle for the RSA problem, and outputs a finite set of short algebraic formulae, one of which splits the integer N.

FACTOR: Given $N \in S$ to compute the list of prime factors of N.

COMPUTE-PHI: Given $N \in S$ to compute $\varphi(N)$.

COMPUTE-LAMBDA: Given $N \in S$ to compute $\lambda(N)$.

RSA-PRIVATE-KEY: Given $(N, e) \in S \times \mathbb{N}$ to output $\perp$ if e is not coprime to $\lambda(N)$, or d such that $ed \equiv 1 \pmod{\lambda(N)}$.

Exercise 24.1.16. Show that RSA $\leq_R$ RSA-PRIVATE-KEY $\leq_R$ COMPUTE-LAMBDA $\leq_R$ FACTOR for integers $N \in \mathbb{N}$.

Exercise 24.1.16 tells us that FACTOR is at least as hard as RSA. A more useful interpretation is that the RSA problem is no harder than factoring. We are interested in the relative difficulty of such problems, as a function of the input size. Lemma 24.1.17 is the main tool for comparing these problems.[5]

Lemma 24.1.17. *Let A be a perfect oracle that takes as input an integer N and outputs a multiple of $\lambda(N)$. Then one can split N in randomised polynomial-time using an expected polynomially many queries to A.*

Proof: Let N be the integer to be factored. We may assume that N is composite, not a prime power, odd and has no very small factors. Let M be the output of the oracle A on N. (Note that the case of non-perfect oracles is not harder: one can easily test whether the output of A is correct by taking a few random integers $1 < a < N$ such that $\gcd(a, N) = 1$ and checking whether $a^M \equiv 1 \pmod{N}$.)

Since N is odd we have that M is even. Write $M = 2^r m$ where m is odd. Now choose uniformly at random an integer $1 < a < N$. Check whether $\gcd(a, N) = 1$. If not then we have split N, otherwise compute $a_0 = a^m \pmod{N}$, $a_1 = a_0^2 \pmod{N}, \ldots, a_r = a^M \pmod{N}$ (this is similar to the Miller-Rabin test; see Section 12.1.2). We know that $a_r = 1$, so either $a_0 = 1$ or else somewhere along the way there is a non-trivial square root x of 1. If $x \neq -1$ then $\gcd(x + 1, N)$ yields a non-trivial factor of N. All computations require a polynomially bounded number of bit operations.

Let p and q be two distinct prime factors of N. Since a is chosen uniformly at random it follows that $\gcd(a, N) > 1$ or $(\frac{a}{p}) = -(\frac{a}{q})$ with probability at least $1/2$. In either case the above process splits N. The expected number of trials to split N is therefore at most 2.

Repeating the above process on each of the factors in turn one can factor N completely. The expected number of iterations is $O(\log(N))$. For a complete anaysis of this reduction see Section 7.7 of Talbot and Welsh [600] or Section 10.6 of Shoup [556]. □

Two special cases of Lemma 24.1.17 are FACTOR $\leq_R$ COMPUTE-LAMBDA and FACTOR $\leq_R$ COMPUTE-PHI. Note that these reductions are randomised and the running time is only an expected value. Coron and May [151] showed a deterministic polynomial time reduction FACTOR $\leq_R$ RSA-PRIVATE-KEY (also see Section 4.6 of [411]).

Exercise 24.1.18. Restricting attention to integers of the form $N = pq$ where p and q are distinct primes, show that FACTOR $\leq_R$ RSA-PRIVATE KEY.

Exercise 24.1.19. Give a more direct (and deterministic) reduction of FACTOR to COMPUTE-PHI for integers of the form $N = pq$ where p and q are distinct primes.

[5]The original RSA paper credits this result to G. Miller.

Exercise 24.1.20.★ Suppose one has a perfect oracle A that takes as input pairs (N, g), where N is an RSA modulus and g is uniformly chosen in $(\mathbb{Z}/N\mathbb{Z})^*$, and returns the order of g modulo N. Show how to use A to factor an RSA modulus N in expected polynomial-time.

Exercise 24.1.21. The **STRONG-RSA** problem is: Given an RSA modulus N and $y \in \mathbb{N}$ to find any pair (x, e) of integers such that $e > 1$ and

$$x^e \equiv y \pmod{N}.$$

Give a reduction from STRONG-RSA to RSA. This shows that the STRONG-RSA problem is not harder than the RSA problem.[6]

We end with some cryptanalysis exercises.

Exercise 24.1.22. Let $N = pq$ be an RSA modulus. Let A be an oracle that takes as input an integer a and returns $a \pmod{\varphi(N)}$. Show how to use A to factor N.

An interesting question is to study the bit security of RSA. More precisely, if (N, e) is an RSA public key one considers an (not necessarily perfect) oracle which computes the least significant bit of x given $y = x^e \pmod{N}$. It can be shown that if one has such an oracle then one can compute x. One approach is to use the binary Euclid algorithm; due to lack of space we simply refer to Alexi, Chor, Goldreich and Schnorr [9] for details of the method and a comprehensive list of references. A simpler approach, which does not use the binary Euclid algorithm, was given by Fischlin and Schnorr [203]. A complete analysis of the security of any bit (not just the least significant bit) was completed by Håstad and Näslund [279].

Exercise 24.1.23. Consider the following variant of RSA encryption. Alice has a public key N and two public exponents e_1, e_2 such that $e_1 \neq e_2$ and $\gcd(e_i, \lambda(N)) = 1$ for $i = 1, 2$. To encrypt to Alice one is supposed to send $\mathsf{c}_1 = \mathsf{m}^{e_1} \pmod{N}$ and $\mathsf{c}_2 = \mathsf{m}^{e_2} \pmod{N}$. Show that if $\gcd(e_1, e_2) = 1$ then an attacker can determine the message given the public key and a ciphertext $(\mathsf{c}_1, \mathsf{c}_2)$.

Exercise 24.1.24. Consider the following signature scheme based on RSA. The public key is an integer $N = pq$, an integer e coprime to $\lambda(N)$ and an integer a such that $\gcd(a, N) = 1$. The private key is the inverse of e modulo $\lambda(N)$, as usual. Let H be a collision-resistant hash function. The signature on a message m is an integer s such that

$$\mathsf{s}^e \equiv a^{H(\mathsf{m})} \pmod{N}$$

where $H(\mathsf{m})$ is interpreted as an integer. Explain how the signer can generate signatures efficiently. Find a known message attack on this system that allows an adversary to make selective forgery of signatures.

24.2 The Textbook Rabin Cryptosystem

The textbook Rabin cryptosystem [494] is given in Figure 24.2. Rabin is essentially RSA with the optimal choice of e, namely $e = 2$.[7] As we will see, the security of Rabin is

[6] The word "strong" is supposed to indicate that the *assumption* that STRONG-RSA is hard is a *stronger assumption* than the assumption that RSA is hard. Of course, the computational problem is weaker than RSA, in the sense that it might be easier to solve STRONG-RSA than RSA.

[7] The original paper [494] proposed encryption as $E_{N,b}(x) = x(x + b) \pmod{N}$ for some integer b. However, there is a gain in efficiency with no loss of security by taking $b = 0$.

more closely related to factoring than RSA. We first have to deal with the problem that if $N = pq$ where p and q are distinct primes then squaring is a four-to-one map (in general) so it is necessary to have a rule to choose the correct solution in decryption.

Lemma 24.2.1. *Suppose p and q are primes such that $p \equiv q \equiv 3 \pmod 4$. Let $N = pq$ and $1 < x < N$ be such that $(\frac{x}{N}) = 1$. Then either x or $N - x$ is a square modulo N.*

Exercise 24.2.2. Prove Lemma 24.2.1.

Definition 24.2.3. Let $p \equiv q \equiv 3 \pmod 4$ be primes. Then $N = pq$ is called a **Blum integer**.

KeyGen(κ): Generate two random $\kappa/2$-bit primes p and q such that $p \equiv q \equiv 3 \pmod 4$ and set $N = pq$. Output the public key $\mathsf{pk} = N$ and the private key $\mathsf{sk} = (p, q)$.

The message space and ciphertext space depend on the redundancy scheme (suppose for the moment that they are $\mathsf{C}_\kappa = \mathsf{M}_\kappa = (\mathbb{Z}/N\mathbb{Z})^*$).

Encrypt(m, N): Compute $\mathsf{c} = \mathsf{m}^2 \pmod N$ (with some redundancy or padding scheme).

Decrypt($\mathsf{c}, (p, q)$): We want to compute $\sqrt{\mathsf{c}} \pmod N$, and this is done by the following method: Compute $\mathsf{m}_p = \mathsf{c}^{(p+1)/4} \pmod p$ and $\mathsf{m}_q = \mathsf{c}^{(q+1)/4} \pmod q$ (see Section 2.9). Test that $\mathsf{m}_p^2 \equiv \mathsf{c} \pmod p$ and $\mathsf{m}_q^2 \equiv \mathsf{c} \pmod q$, and if not then output $\perp$. Use the Chinese remainder theorem (Exercise 2.6.3) to obtain four possibilities for $\mathsf{m} \pmod N$ such that $\mathsf{m} \equiv \pm\mathsf{m}_p \pmod p$ and $\mathsf{m} \equiv \pm\mathsf{m}_q \pmod q$. Use the redundancy (see later) to determine the correct value m and return $\perp$ if there is no such value.

Sign($\mathsf{m}, (p, q)$): Ensure that $(\frac{\mathsf{m}}{N}) = 1$ (possibly by adding some randomness). Then either m or $N - \mathsf{m}$ is a square modulo N. Compute $\mathsf{s} = \sqrt{\pm\mathsf{m}} \pmod N$ by computing $(\pm\mathsf{m})^{(p+1)/4} \pmod p$, $(\pm\mathsf{m})^{(q+1)/4} \pmod q$ and applying the Chinese remainder theorem.

Verify($\mathsf{m}, \mathsf{s}, N$): Check whether $\mathsf{m} \equiv \pm\mathsf{s}^2 \pmod N$.

Figure 24.2: Textbook Rabin.

Note that, as with RSA, the value m in encryption is actually a symmetric key (passed through a padding scheme) while in signing it is a hash of the message. The choice of $p, q \equiv 3 \pmod 4$ is to simplify the taking of square roots (and is also used in the redundancy schemes below); the Rabin scheme can be used with other moduli.

24.2.1 Redundancy Schemes for Unique Decryption

To ensure that decryption returns the correct message it is necessary to have some redundancy in the message, or else to send some extra bits. We now describe three solutions to this problem.

- **Redundancy in the message for Rabin**: For example, insist that the least significant l bits (where $l > 2$ is some known parameter) of the binary string m are all ones. (Note 8.14 of [418] suggests repeating the last l bits of the message.) If l is big enough then it is unlikely that two different choices of square root would have the right pattern in the l bits.

A message m is encoded as $x = 2^l\mathsf{m} + (2^l - 1)$, and so the message space is $\mathsf{M}_\kappa = \{\mathsf{m} : 1 \leq \mathsf{m} < N/2^l, \gcd(N, 2^l\mathsf{m} + (2^l - 1)) = 1\}$ (alternatively, $\mathsf{M}_\kappa = \{0,1\}^{\kappa-l-2}$). The ciphertext is $\mathsf{c} = x^2 \pmod N$. Decryption involves computing the four square roots of c. If none, or more than one, of the roots has all l least significant bits equal to one and so corresponds to an element of M_κ then decryption fails (return $\perp$). Otherwise output the message $\mathsf{m} = \lfloor x/2^l \rfloor$.

This method is a natural choice, since some padding schemes for CCA security (such as OAEP) already have sections of the message with a fixed pattern of bits.

Note that, since N is odd, the least significant bit of $N - x$ is different to the least significant bit of x. Hence, the $l \geq 1$ least significant bits of x and $N - x$ are never equal. Treating the other two square roots of $x^2 \pmod N$ as random integers it is natural to conjecture that the probability that either of them has a specific pattern of their l least significant bits is roughly $2/2^l$. This conjecture is confirmed by experimental evidence. Hence, the probability of decryption failure is approximately $1/2^{l-1}$.

- **Extra bits for Rabin**: Send two extra bits of information to specify the square root. For example, one could send the value $b_1 = (\frac{\mathsf{m}}{N})$ of the Jacobi symbol (the set $\{-1, 1\}$ can be encoded as a bit under the map $x \mapsto (x+1)/2$), together with the least significant bit b_2 of the message. The ciphertext space is now $\mathsf{C}_\kappa = (\mathbb{Z}/N\mathbb{Z})^* \times \{0,1\}^2$ and, for simplicity of exposition, we take $\mathsf{M}_\kappa = (\mathbb{Z}/N\mathbb{Z})^*$.

  These two bits allow unique decryption, since $(\frac{-1}{N}) = 1$, m and $N - \mathsf{m}$ have the same Jacobi symbol, and if m is odd then $N - \mathsf{m}$ is even.

  Indeed, when using the Chinese remainder theorem to compute square roots then one computes m_p and m_q such that $(\frac{\mathsf{m}_p}{p}) = (\frac{\mathsf{m}_q}{q}) = 1$. Then decryption using the bits b_1, b_2 is: if $b_1 = 1$ then the decryption is $\pm CRT(\mathsf{m}_p, \mathsf{m}_q)$ and if $b_1 = -1$ then solution is $\pm CRT(-\mathsf{m}_p, \mathsf{m}_q)$.

  This scheme is close to optimal in terms of ciphertext expansion (though see Exercise 24.2.6 for an improvement) and decryption never fails. The drawbacks are that the ciphertext contains some information about the message (and so the scheme is not IND-CPA secure), and encryption involves computing the Jacobi symbol, which typically requires far more computational resources than the single squaring modulo N.

- **Williams**: Let $N = pq$ where $p, q \equiv 3 \pmod 4$. If $p \not\equiv \pm q \pmod 8$ then $(\frac{2}{N}) = -1$. Hence, for every $1 \leq x < N$ exactly one of x, $N - x$, $2x$, $N - 2x$ is a square modulo N (see Exercise 24.6.3). Without loss of generality we therefore assume that $p \equiv 3 \pmod 8$ and $q \equiv 7 \pmod 8$. The integer N is called a **Williams integer** in this situation.

  Williams [633] suggests encoding a message $1 \leq \mathsf{m} < N/8 - 1$ (alternatively, $\mathsf{m} \in \mathsf{M}_\kappa = \{0,1\}^{\kappa-5}$) as an integer x such that x is even and $(\frac{x}{N}) = 1$ (and so x or $-x$ is a square modulo N) by

$$x = P(\mathsf{m}) = \begin{cases} 4(2\mathsf{m}+1) & \text{if } (\frac{2\mathsf{m}+1}{N}) = 1, \\ 2(2\mathsf{m}+1) & \text{if } (\frac{2\mathsf{m}+1}{N}) = -1 \end{cases}$$

  The encryption of m is then $\mathsf{c} = P(\mathsf{m})^2 \pmod N$. One has $\mathsf{C}_\kappa = (\mathbb{Z}/N\mathbb{Z})^*$.

  To decrypt one computes square roots to obtain the unique even integer $1 < x < N$ such that $(\frac{x}{N}) = 1$ and $x^2 \equiv \mathsf{c} \pmod N$. If $8 \mid x$ then decryption fails (return $\perp$). Otherwise, return $\mathsf{m} = (x/2 - 1)/2$ if $x \equiv 2 \pmod 4$ and $\mathsf{m} = (x/4 - 1)/2$ if $x \equiv 0 \pmod 4$.

Unlike the extra bits scheme, this does not reveal information about the ciphertext. It is almost optimal from the point of view of ciphertext expansion. But it still requires the encrypter to compute a Jacobi symbol (hence losing the performance advantage of Rabin over RSA). The Rabin cryptosystem with the Williams padding is sometimes called the **Rabin-Williams cryptosystem**.

Exercise 24.2.4. Prove all the unproved claims in the above discussion of the Williams redundancy scheme.

Exercise 24.2.5. Let $N = (2^{59} + 21)(2^{20} + 7)$. The three ciphertexts below are Rabin encryptions for each of the three redundancy schemes above (in the first case, $l = 5$). Determine the corresponding message in each case.

$$273067682422\,, \quad (309135051204, -1, 0)\,, \quad 17521752799.$$

Exercise 24.2.6. (Freeman-Goldreich-Kiltz-Rosen-Segev [211]) Let N be a Williams integer. This is a variant of the "extra bits" method. Let $u_1 = -1$ and $u_2 = 2$. To encrypt message $\mathsf{m} \in (\mathbb{Z}/N\mathbb{Z})^*$ one first determines the bits b_1 and b_2 of the "extra bits" redundancy scheme (i.e., $b_1 = 1$ if and only if $(\frac{\mathsf{m}}{N}) = +1$ and b_2 is the least significant bit of m). Compute the ciphertext

$$\mathsf{c} = \mathsf{m}^2 u_1^{1-b_1} u_2^{b_2} \pmod{N}.$$

Show how a user who knows p and q can decrypt the ciphertext. Show that this scheme still leaks the least significant bit of m (and hence is still not IND-CPA secure), but no longer leaks $(\frac{\mathsf{m}}{N})$.

24.2.2 Variants of Rabin

In terms of computational performance, Rabin encryption is extremely fast (as long as encryption does not require computing a Jacobi symbol) while decryption, using the Chinese remainder theorem, is roughly the same speed as RSA decryption.

Exercise 24.2.7. Describe and analyse the Rabin cryptosystem using moduli of the form $N = pqr$ where p, q and r are distinct primes. What are the advantages and disadvantages of Rabin in this setting?

Exercise 24.2.8. (Takagi-Rabin) Describe and analyse the Rabin cryptosystem using moduli of the form $N = p^r q^s$ $(r \neq s)$. Is there any advantage from using Rabin in this setting?

We now discuss compression of Rabin signatures. For further discussion of these ideas, and an alternative method, see Gentry [252].

Example 24.2.9. (Bleichenbacher [68]) Suppose s is a Rabin signature on a message m, so that $\mathsf{s}^2 \equiv \pm H(\mathsf{m}) \pmod{N}$. To compress s to half the size one uses the Euclidean algorithm on (s, N) to compute a sequence of values $r_i, u_i, v_i \in \mathbb{Z}$ such that $r_i = u_i\mathsf{s} + v_i N$. Let i be the index such that $|r_i| < \sqrt{N} < |r_{i-1}|$. Then $r_i \equiv u_i \mathsf{s} \pmod{N}$ and so

$$r_i^2 \equiv u_i^2 \mathsf{s}^2 \equiv \pm u_i^2 H(\mathsf{m}) \pmod{N}.$$

One can therefore send u_i as the signature. Verification is to compute $w = \pm u_i^2 H(\mathsf{m}) \pmod{N}$ and check that w is a perfect square in $\mathbb{Z}$ (e.g., using the method of Exercise 2.4.9 or Exercise 2.2.8). Part 6 of Lemma 2.3.3 states $|r_{i-1}u_i| \leq N$ and so $|u_i| < \sqrt{N}$. Hence, this approach compresses the signature to half the size.

Example 24.2.10. (Bernstein; Coron and Naccache) Another way to compress Rabin signatures is to send the top half of the bits of s. In other words, the signature is $\mathsf{s}' = \lfloor \mathsf{s}/2^{\kappa/2} \rfloor$ if $N < 2^\kappa$. To verify s' one uses Coppersmith's method to find the small solution x to the equation

$$(\mathsf{s}'2^{\kappa/2} + x)^2 \pm H(\mathsf{m}) \equiv 0 \pmod{N}.$$

Verification of this signature is much slower than the method of Example 24.2.9.

24.2.3 Security of Textbook Rabin

Since the Rabin cryptosystem involves squaring it is natural to assume the security is related to computing square roots modulo N, which in turn is equivalent to factoring. Hence, an important feature of Rabin compared with RSA is that the hardness of breaking Rabin can be shown to be equivalent to factoring.

Definition 24.2.11. Let $S = \mathbb{N}$ or $S = \{pq : p, q \equiv 3 \pmod 4, \text{ primes}\}$. The computational problem **SQRT-MOD-N** is: Given $N \in S$ and $y \in \mathbb{Z}/N\mathbb{Z}$ to output $\perp$ if y is not a square modulo N, or a solution x to $x^2 \equiv y \pmod N$.

Lemma 24.2.12. *SQRT-MOD-N is equivalent to FACTOR.*

Proof: Suppose we have a FACTOR oracle and are given a pair (N, y). Then one can use the oracle to factor N and then solve SQRT-MOD-N using square roots modulo p and Hensel lifting and the Chinese remainder theorem. This reduction is polynomial-time.

Conversely, suppose we have a SQRT-MOD-N oracle A and let N be given. First, if $N = p^e$ then we can factor N in polynomial time (see Exercise 2.2.9). Hence we may now assume that N has at least two distinct prime factors.

Choose a random $x \in \mathbb{Z}_N^*$ and set $y = x^2 \pmod N$. Call A on y to get x'. We have $x^2 \equiv (x')^2 \pmod N$ and there are at least four possible solutions x'. All but two of these solutions will give a non-trivial value of $\gcd(x - x', N)$. Hence, since x was chosen randomly, there is probability at least $1/2$ that we can split N. Repeating this process splits N (the expected number of trials is at most 2). As in Lemma 24.1.17 one can repeat the process to factor N in $O(\log(N))$ iterations. The entire reduction is therefore polynomial-time. □

An important remark about the above proof is that the oracle A is not assumed to output a random square root x' of y. Indeed, A could be deterministic. The randomness comes from the choice of x in the reduction.

Exercise 24.2.13. Consider the computational problem FOURTH-ROOT: Given $y \in \mathbb{Z}_N^*$ compute a solution to $x^4 \equiv y \pmod N$ if such a solution exists. Give reductions that show that FOURTH-ROOT is equivalent to FACTOR in the case $N = pq$ with p, q distinct odd primes.

It is intuitively clear that any algorithm that breaks the one-way encryption property (or selective signature forgery) of Rabin under passive attacks must compute square roots modulo N. We have seen that SQRT-MOD-N is equivalent to FACTOR. Thus we expect breaking Rabin under passive atacks to be as hard as factoring. However, giving a precise security proof involves taking care of the redundancy scheme.

Theorem 24.2.14. *Let $N = pq$, where $p \equiv q \equiv 3 \pmod 4$ are primes, and define $S_{N,l} = \{1 \leq x < N : \gcd(x, N) = 1, 2^l \mid (x + 1)\}$. Assume the probability, over $x \in \mathbb{Z}_N^* - S_{N,l}$, that there exists $y \in S_{N,l}$ with $x \neq y$ but $x^2 \equiv y^2 \pmod N$, is $1/2^{l-1}$. Then breaking*

the one-way encryption security property of the Rabin cryptosystem with the "redundancy in the message" redundancy scheme where $l = O(\log(\log(N)))$ under passive attacks is equivalent to factoring Blum integers.

Theorem 24.2.15. *Breaking the one-way encryption security property of the Rabin cryptosystem with the "extra bits" redundancy scheme under passive attacks is equivalent to factoring products $N = pq$ of primes $p \equiv q \equiv 3 \pmod 4$.*

Theorem 24.2.16. *Breaking the one-way encryption security property of the Rabin cryptosystem with the Williams redundancy scheme under passive attacks is equivalent to factoring products $N = pq$ of primes $p \equiv q \equiv 3 \pmod 4$, $p \not\equiv \pm q \pmod 8$.*

Note that Theorem 24.2.14 gives a strong security guarantee when l is small, but in that case decryption failures are frequent. Indeed, there is no choice of l for the Rabin scheme with redundancy in the message that provides both a tight reduction to factoring and negligible probability of decryption failure.

We prove the first and third of these theorems and leave Theorem 24.2.15 as an exercise.

Proof: (Theorem 24.2.14) Let A be an oracle that takes a Rabin public key N and a ciphertext c (with respect to the "redundancy in the message" padding scheme) and returns either the corresponding message m or an invalid ciphertext symbol $\perp$.

Choose a random $x \in \mathbb{Z}_N^*$ such that neither x nor $N-x$ satisfy the redundancy scheme (i.e., the l least significant bits are not all 1). Set $\mathsf{c} = x^2 \pmod N$ and call the oracle A on c. The oracle A answers with either a message m or $\perp$.

According to the (heuristic) assumption in the theorem, the probability that exactly one of the two (unknown) square roots of c modulo N has the correct l least significant bits is $2^{-(l-1)}$. If this is the case then calling the oracle A on c will output a value m such that, writing $x' = 2^l\mathsf{m} + (2^l - 1)$, we have $(x')^2 \equiv x^2 \pmod N$ and $x' \not\equiv \pm x \pmod N$. Hence $\gcd(x' - x, N)$ will split N.

We expect to require approximately 2^{l-1} trials before factoring N with this method. Hence, the reduction is polynomial-time if $l = O(\log(\log(N)))$. □

Proof: (Proof of Theorem 24.2.16; following Williams [633]) Let A be an oracle that takes a Rabin public key N and a ciphertext c (with respect to the Williams redundancy scheme) and returns either the corresponding message m or an invalid ciphertext symbol $\perp$.

Choose a random integer x such that $(\frac{x}{N}) = -1$, e.g., let $x = \pm 2z^2 \pmod N$ for random $z \in (\mathbb{Z}/N\mathbb{Z})^*$. Set $\mathsf{c} = x^2 \pmod N$ and call A on (N, c). The oracle computes the unique even integer $1 < x' < N$ such that $(x')^2 \equiv \mathsf{c} \pmod N$ and $(\frac{x'}{N}) = 1$. The oracle then attempts to decode x' to obtain the message. If $8 \nmid x'$ (which happens with probability 3/4) then decoding succeeds and the corresponding message m is output by the oracle. Given m we can recover the value x' as $2(2\mathsf{m}+1)$ or $4(2\mathsf{m}+1)$, depending on the value of $(\frac{2\mathsf{m}+1}{N})$, and then factor N as $\gcd(x' - x, N)$.

If $8 \mid x'$ then the oracle outputs $\perp$ so we compute $\mathsf{c}' = \mathsf{c}2^{-4} \pmod N$ and call the oracle on c'. The even integer x'' computed by the oracle is equal to $x'/4$ and so the above argument may apply. In extremely rare cases one might have to repeat the process $\frac{1}{2}\log_2(N)$ times, but the expected number of trials is constant. □

Exercise 24.2.17. Prove Theorem 24.2.15.

Exercise 24.2.18. Prove Theorem 24.2.14 when the message space is $\{0,1\}^{\kappa-l-2}$.

The above theorems show that the hardness guarantee for the Rabin cryptosystem is often stronger than for the RSA cryptosystem (at least, under passive attacks). Hence

the Rabin cryptosystem is very attractive: it has faster public operations and also has a stronger security guarantee than RSA. On the other hand, the ideas used in the proofs of these theorems can also be used to give adaptive (CCA) attacks on the Rabin scheme that allow the attacker to determine the private key (i.e., the factorisation of the modulus). In comparison, a CCA attack on textbook RSA only decrypts a single message rather than computes the private key.

Example 24.2.19. We describe a CCA attacker giving a total break of Rabin with "redundancy in the message".

As in the proof of Theorem 24.2.14 the adversary chooses a random $x \in \mathbb{Z}_N^*$ such that neither x nor $N-x$ satisfy the redundancy scheme (i.e., the l least significant bits are not all 1). Set $\mathsf{c} = x^2 \pmod N$ and call the decryption oracle on c. The oracle answers with either a message m or $\perp$. Given m one computes x' such that $\gcd(x'-x, N)$ splits N.

Exercise 24.2.20. Give CCA attacks giving a total break of Rabin when using the other two redundancy schemes ("extra bits" and Williams).

As we have seen, the method to prove that Rabin encryption has one-way security under a passive attack is also the method to give a CCA attack on Rabin encryption. It was remarked by Williams [633] that such a phenomenon seems to be inevitable. This remark has been formalised and discussed in detail by Paillier and Villar [477].

Exercise 24.2.21. Generalise Rabin encryption to $N = pq$ where $p \equiv q \equiv 1 \pmod 3$ and encryption is $\mathsf{c} = \mathsf{m}^3 \pmod N$. How can one specify redundancy? Is the security related to factoring in this case?

Exercise 24.2.22. Consider the following public key cryptosystem related to Rabin: A user's public key is a product $N = pq$ where p and q are primes congruent to 3 modulo 4. To encrypt a message $1 < \mathsf{m} < N$ to the user compute and send

$$\mathsf{c}_1 = \mathsf{m}^2 \pmod N \qquad \text{and} \qquad \mathsf{c}_2 = (\mathsf{m}+1)^2 \pmod N.$$

Show that if $x^2 \equiv y^2 \pmod N$ and $(x+1)^2 \equiv (y+1)^2 \pmod N$ then $x \equiv y \pmod N$. Hence show that decryption is well-defined.

Show that this cryptosystem does not have OWE security under a passive attack.

24.2.4 Other Computational Problems Related to Factoring

We now give some other computational problems in algebraic groups modulo N that are related to factoring.

Exercise 24.2.23. Let $N = pq \in \mathbb{N}$ be a product of two large primes $p \equiv q \equiv 3 \pmod 4$ and let $G = \{x^2 : x \in (\mathbb{Z}/N\mathbb{Z})^*\}$. Let A be an oracle for CDH in G (i.e., $A(g, g^a, g^b) = g^{ab}$). Show how to use A to factor N.

Exercise 24.2.24. Let $N = pq$. Show how to factor N when given $M = (p+1)(q+1)$.

More generally, given $N = pq$ and $M = \Phi_k(p)\Phi_k(q)$ one can split N as follows: Write $F_1(x,y) = xy - N$ and $F_2(x,y) = \Phi_k(x)\Phi_k(y) - M$. One then takes the resultant of $F_1(x,y)$ and $F_2(x,y)$ to get a polynomial $G(x)$. Note that $G(x)$ has p as a root, so one can find p by taking real roots of $G(x)$ to high precision.

Exercise 24.2.25. Let $N = pq = 1125907426181141$ and $M = (p^2+p+1)(q^2+q+1) = 1267668742445499725931290297061$. Determine p and q using resultants as above.

Exercise 24.2.26. Let $N = pq$ where $p \equiv q \equiv 1 \pmod 4$ are primes. Recall that the torus $\mathbb{T}_2(\mathbb{Z}/N\mathbb{Z})$ has order $(p+1)(q+1)$. Let $G = \{g^2 : g \in \mathbb{T}_2(\mathbb{Z}/N\mathbb{Z})\}$. Let A be an oracle for CDH in G. Use the method of Exercise 24.2.23 to factor N using A.

Exercise 24.2.27. Let $N = pq$ where p and q are odd primes and let $E : y^2 = x^3 + a_4x + a_6$ be an elliptic curve. Suppose A is an oracle that, on input $(P, [a]P, [b]P)$, where P has odd order, outputs $[ab]P$ in $\langle P \rangle$. Explain why one can not immediately factor N using this oracle. Consider now an oracle A, taking input $(a_4, a_6, P, [a]P, [b]P)$ where P lies on the elliptic curve $y^2 = x^3 + a_4x + a_6$ modulo N and P has odd order, that outputs $[ab]P$. Show how to use A to factor N.

There are two approaches to using information about $\#E(\mathbb{Z}/N\mathbb{Z})$ to split N. One is more suitable when one has an oracle that computes $\#E(\mathbb{Z}/N\mathbb{Z})$ and the other is more suitable when E is fixed.

Example 24.2.28. (Kunihiro and Koyama [358]) Let $N = pq$ be a product of two primes. Let A be an oracle that takes as input (N, a_4, a_6) and returns $M = \#E(\mathbb{Z}/N\mathbb{Z})$ where $E : y^2 = x^3 + a_4x + a_6$.

Given the oracle A one can split N using exactly the same method as Lemma 24.1.17. First choose a random elliptic curve E together with a point P on it modulo N. Use the oracle A to compute $M = \#E(\mathbb{Z}/N\mathbb{Z})$. Now, find small prime factors l of M (such as $l = 2$) and compute $[M/l]P = (x : y : z)$ in projective coordinates. There is a good chance that l divides both $\#E(\mathbb{F}_p)$ and $\#E(\mathbb{F}_q)$ and that $P \pmod p$ has order divisible by l but $P \pmod q$ does not. Hence, $\gcd(z, N)$ splits N.

Exercise 24.2.29. (Martín Molleví, Morillo and Villar [400]) Use the method of Example 24.2.28 to show how to factor N given an oracle A that takes as input (N, a_4, a_6, x_P, y_P) and returns the order of the point $P = (x_P, y_P) \in E(\mathbb{Z}/N\mathbb{Z})$.

Example 24.2.30. Let $N = pq$ be a product of two primes. Let $E : y^2 = x^3 + a_4x + a_6$ be an elliptic curve modulo N such that E is not supersingular modulo p or q. Let $M = \#E(\mathbb{Z}/N\mathbb{Z})$ be given.

Now choose a random integer $1 \leq x_P < N$. There may not be a point on $E(\mathbb{Z}/N\mathbb{Z})$ with x-coordinate x_P. Indeed, we hope that there is not. Then there is a quadratic twist $E' : uY^2 = X^3 + a_4X + a_6$ of E with a point $P = (x_P, y_P) \in E'(\mathbb{Z}/N\mathbb{Z})$. With probability $1/2$ we have $\#E(\mathbb{F}_p) = \#E'(\mathbb{F}_p)$ but $\#E(\mathbb{F}_p) \neq \#E'(\mathbb{F}_p)$ (or vice versa). It is not necessary to compute y_P or to determine E'. Using x-coordinate only arithmetic on E one can compute the projective representation $(x_Q : z_Q)$ for the x-coordinate of $Q = [M](x_P, y_P)$ on E'. Then $\gcd(z_Q, N)$ splits N.

Exercise 24.2.31. Adapt the methods in Examples 24.2.28 and 24.2.30 to give alternative methods to factor $N = pq$ given $\#\mathbb{T}_2(\mathbb{Z}/N\mathbb{Z})$ or $\#\mathbb{T}_6(\mathbb{Z}/N\mathbb{Z})$.

24.3 Homomorphic Encryption

Homomorphic encryption was defined in Section 23.3.1. We first remark that the textbook RSA scheme is homomorphic for multiplication modulo N: If $\mathsf{c}_1 \equiv \mathsf{m}_1^e \pmod N$ and $\mathsf{c}_2 \equiv \mathsf{m}_2^e \pmod N$ then $\mathsf{c}_1\mathsf{c}_2 \equiv (\mathsf{m}_1\mathsf{m}_2)^e \pmod N$. Indeed, this property is behind the CCA attack on textbook RSA encryption. Padding schemes can destroy this homomorphic feature.

Exercise 24.3.1. Show that textbook Rabin encryption is not homomorphic for multiplication when using any of the redundancy schemes of Section 24.2.1.

We now give a scheme that is homomorphic for addition, and that allows a much larger range of values for the message compared with the scheme in Exercise 23.3.5.

Example 24.3.2. (Paillier [474]) Let $N = pq$ be a user's public key. To encrypt a message $\mathsf{m} \in \mathbb{Z}/N\mathbb{Z}$ to the user choose a random integer $1 < u < N$ (note that, with overwhelming probability, $u \in (\mathbb{Z}/N\mathbb{Z})^*$) and compute the ciphertext

$$\mathsf{c} = (1 + N\mathsf{m})u^N \pmod{N^2}.$$

To decrypt compute

$$\mathsf{c}^{\lambda(N)} \equiv 1 + \lambda(N)N\mathsf{m} \pmod{N^2}$$

and hence determine $\mathsf{m} \pmod N$ (this requires multiplication by $\lambda(N)^{-1} \pmod N$).

The homomorphic property is: if c_1 and c_2 are ciphertexts encrypting m_1 and m_2 respectively, then

$$\mathsf{c}_1\mathsf{c}_2 \equiv (1 + N(\mathsf{m}_1 + \mathsf{m}_2))(u_1u_2)^N \pmod{N^2}$$

encrypts $\mathsf{m}_1 + \mathsf{m}_2 \pmod N$.

Exercise 24.3.3. Verify the calculations in Example 24.3.2.

As always, one cannot obtain CCA secure encryption using a homomorphic scheme. Hence, one is only interested in passive attacks. To check whether or not a Paillier ciphertext c corresponds to a specific message m is precisely solving the following computational problem.

Definition 24.3.4. Let $N = pq$. The **composite residuosity problem** is: Given $y \in \mathbb{Z}/N^2\mathbb{Z}$ to determine whether or not $y \equiv u^N \pmod{N^2}$ for some $1 < u < N$.

Exercise 24.3.5. Show that the Paillier encryption scheme has IND-CPA security if and only if the composite residuosity problem is hard.

Exercise 24.3.6. Show that composite residuosity is not harder than factoring.

Exercise 24.3.7. Show how to use the Chinese remainder theorem to speed up Paillier decryption.

Encryption using the Paillier scheme is rather slow, since one needs an exponentiation to the power N modulo N^2. One can use sliding windows for this exponentiation, though since N is fixed one might prefer to use an addition chain optimised for N. Exercises 24.3.8 and 24.3.9 suggest variants with faster encryption. The disadvantage of the scheme in Exercise 24.3.8 is that it requires a different computational assumption. The disadvantage of the scheme in Exercise 24.3.9 is that it is no longer homomorphic.

Exercise 24.3.8.★ Consider the following efficient variant of the Paillier cryptosystem. The public key of a user consists of N and an integer $h = u^N \pmod{N^2}$ where $1 < u < N$ is chosen uniformly at random. To encrypt a message m to the user, choose a random integer $0 \leq x < 2^k$ (e.g., with $k = 256$) and set

$$\mathsf{c} \equiv (1 + N\mathsf{m})h^x \pmod{N^2}.$$

State the computational assumption underlying the IND-CPA security of the scheme. Give an algorithm to break the IND-CPA security that requires $O(2^{k/2})$ multiplications modulo N^2. Use multi-exponentiation to give an even more efficient variant of the Paillier cryptosystem, at the cost of even larger public keys.

Exercise 24.3.9. (Catalano, Gennaro, Howgrave-Graham, Nguyen [125]) A version of the Paillier cryptosystem for which encryption is very efficient is the following: The public key is (N, e) such that $\gcd(e, N\lambda(N)) = 1$. One thinks of e as being small. To encrypt one chooses a random integer $1 < u < N$ and computes

$$\mathsf{c} = (1 + N\mathsf{m})u^e \pmod{N^2}.$$

Decryption begins by performing RSA decryption of c modulo N to obtain u.

Write down the decryption algorithm for this system. Explain why this encryption scheme is no longer homomorphic.

Exercise 24.3.10. Consider the following variant of the Paillier cryptosystem: The public key consists of $N = pq$ and an integer g such that $g^{\lambda(N)} \equiv 1 + N \pmod{N^2}$. To encrypt a message $0 \leq \mathsf{m} < N$ compute $g^{\mathsf{m}}u^N \pmod{N^2}$ where $1 < u < N$ is random.

Give key generation and decryption algorithms for this cryptosystem. Give a chosen ciphertext attack on this cryptosystem that reveals the private key.[8]

Exercise 24.3.11. (Damgård-Jurik [165]) Generalise the Paillier cryptosystem so the message space is $\mathbb{Z}/N^k\mathbb{Z}$. Explain how to decrypt.

Exercise 24.3.12. (Okamoto-Uchiyama [471]) Let $N = p^2q$ where p and q are distinct primes of similar size such that $p \nmid (q-1)$. Choose a random element $1 < u < N$ (note that, with overwhelming probability, $u \in (\mathbb{Z}/N\mathbb{Z})^*$) and set $g = u^N(1-p) \pmod{N}$. The public key of the **Okamoto-Uchiyama scheme** is (N, g). To encrypt a message $\mathsf{m} \in \mathbb{Z}/p\mathbb{Z}$ (in practice, since p is not known, one would assume $0 \leq \mathsf{m} < N^{1/3}$) one chooses a random element $1 < u < N$ and computes

$$\mathsf{c} = g^{\mathsf{m}}u^N \pmod{N}.$$

To decrypt one computes $((\mathsf{c}^{p-1} \pmod{p^2}) - 1)/p$.

Show that $g^{p-1} \equiv 1 + p \pmod{p^2}$ and hence that decryption does compute m. Show that the scheme is homomorphic with respect to addition modulo p. Define the computational problem underlying the IND-CPA security of this scheme.

Show that if one has access to a decryption oracle then one can determine the factorisation of N.

24.4 Algebraic Attacks on Textbook RSA and Rabin

The goal of this section is to briefly describe a number of relatively straightforward attacks on the textbook RSA and Rabin cryptosystems. These attacks can all be prevented if one uses a sufficiently good padding scheme. Indeed, by studying these attacks one develops a better idea of what properties are required of a padding scheme.

24.4.1 The Håstad Attack

We now present an attack that can be mounted on the RSA or Rabin schemes in a multi-user situation. Note that such attacks are not covered by the standard security model for encryption as presented in Chapter 1.

[8] This scheme was proposed by Choi, Choi and Won at ICISC 2001 and an attack was given by Sakurai and Takagi at ACISP 2002.

Example 24.4.1. Suppose three users have RSA public keys N_1, N_2, N_3 and all use encryption exponent $e = 3$. Let $0 < \mathsf{m} < \min\{N_1, N_2, N_3\}$ be a message. If m is encrypted to all three users then an attacker can determine m from the three ciphertexts $\mathsf{c}_1, \mathsf{c}_2$ and c_3 as follows: The attacker uses the Chinese remainder theorem to compute $1 < \mathsf{c} < N_1N_2N_3$ such that $\mathsf{c} \equiv \mathsf{m}^3 \pmod{N_i}$ for $1 \leq i \leq 3$. It follows that $\mathsf{c} = \mathsf{m}^3$ over $\mathbb{Z}$ and so one can determine m using root finding algorithms.

This attack is easily prevented by using randomised padding schemes (assuming that the encryptor is not so lazy that they re-use the same randomness each time). Nevertheless, this attack seems to be one of the reasons why modern systems use $e = 65537 = 2^{16}+1$ instead of $e = 3$.

Exercise 24.4.2. Show that the Håstad attack applies when the same message is sent using textbook Rabin encryption (with any of the three redundancy schemes) to two users.

Exercise 24.4.3. Two users have Rabin public keys $N_1 = 144946313$ and $N_2 = 138951937$. The same message m is encrypted using the "extra bits" padding scheme to the two users, giving ciphertexts

$$C_1 = (48806038, -1, 1) \text{ and } C_2 = (14277753, -1, 1).$$

Use the Håstad attack to find the corresponding message.

24.4.2 Algebraic Attacks

We already discussed a number of easy algebraic attacks on textbook RSA, all of which boil down to exploiting the multiplicative property

$$\mathsf{m}_1^e \mathsf{m}_2^e \equiv (\mathsf{m}_1\mathsf{m}_2)^e \pmod{N}.$$

We also noted that, since textbook RSA is deterministic, it can be attacked by trying all messages. Hence, if one knows that $1 \leq \mathsf{m} < 2^k$ (for example, if m is a k-bit symmetric key) then one can attack the system in at most 2^k exponentiations modulo N. We now show that one can improve this to roughly $\sqrt{2^k}$ exponentiations in many cases.

Exercise 24.4.4. (Boneh, Joux, Nguyen [82]) Suppose $\mathsf{c} = \mathsf{m}^e \pmod{N}$ where $1 \leq \mathsf{m} < 2^k$. Show that if $\mathsf{m} = \mathsf{m}_1\mathsf{m}_2$ for two integers $1 < \mathsf{m}_1, \mathsf{m}_2 < B$ then one can determine m in $O(B)$ exponentiations modulo N. If $B = 2^{k/2+\epsilon}$ then the probability that m splits in this way is noticeable.

24.4.3 Desmedt-Odlyzko Attack

This is a "lunchtime attack", proposed by Desmedt and Odlyzko in [170], on textbook RSA signatures. It can produce more forgeries than calls to the signing oracle. The basic idea is to query the signing oracle on the first r prime numbers $p_1, \ldots, p_r$ to get signatures $\mathsf{s}_1, \ldots, \mathsf{s}_r$. Then, for any message m, if m is a product of powers of the first r primes $\mathsf{m} = \prod_{i=1}^r p_i^{f_i}$ then the corresponding signature is

$$\mathsf{s} = \prod_{i=1}^r \mathsf{s}_i^{f_i}.$$

This attack is not feasible if messages are random elements between 1 and N (as the probability of smoothness is usually negligible) but it can be effective if messages in the system are rather small.

Exercise 24.4.5. Let $N = 9178628368309$ and $e = 7$ be an RSA public key. Suppose one learns that the signatures of 2, 3 and 5 are 872240067492, 6442782604386 and 1813566093366 respectively. Determine the signatures for messages $\mathsf{m} = 6, 15, 12$ and 100.

An analogous attack applies to encryption: Ask for decryptions of the first r primes (treating them as ciphertexts) and then, given a challenge ciphertext c, if $\mathsf{c} \equiv \prod_{i=1}^{r} p_i^{e_i}$ then one can work out the decryption of c. Since ciphertexts (even of small messages) are of size up to N this attack is usually not faster than factoring the modulus.

This idea, together with a number of other techniques, has been used by Coron, Naccache, Tibouchi and Weinmann [153] to attack real-world signature proposals.

24.4.4 Related Message Attacks

This attack is due to Franklin and Reiter.[9] Consider textbook RSA with small exponent e or textbook Rabin ($e = 2$). Suppose we obtain ciphertexts c_1 and c_2 (with respect to the same public key (N, e)) for messages m and $\mathsf{m} + a$ for some known integer a. Then m is a common root modulo N of the two polynomials $F_1(x) = x^e - \mathsf{c}_1$ and $F_2(x) = (x+a)^e - \mathsf{c}_2$ (in the case of Rabin we may have polynomials like $F_1(x) = (2^l(x+1) - 1)^2 - \mathsf{c}_1$ or $F_1(x) = (2(2x+1))^2 - \mathsf{c}_1$). Hence one can run Euclid's algorithm on $F_1(x)$ and $F_2(x)$ in $(\mathbb{Z}/N\mathbb{Z})[x]$ and this will either lead to a factor of N (since performing polynomial division in $(\mathbb{Z}/N\mathbb{Z})[x]$ involves computing inverses modulo N) or will output, with high probability, a linear polynomial $G(x) = x - \mathsf{m}$.

Euclid's algorithm for polynomials of degree e has complexity $O(e^2 M(\log(N)))$ or $O(M(e)\log(e)M(\log(N)))$ bit operations. Hence, this method is feasible only when e is rather small (e.g., $e < 2^{30}$).

Exercise 24.4.6. Extend the Franklin-Reiter attack to ciphertexts c_1 and c_2 (again, for the same public key) where c_1 is an encrypion of m and c_2 is an encryption of $a\mathsf{m} + b$ for known integers a and b.

Exercise 24.4.7. Let $N = 2157212598407$ and $e = 3$. Suppose we have ciphertexts

$$\mathsf{c}_1 = 1429779991932 \quad \text{and} \quad \mathsf{c}_2 = 655688908482$$

such that c_1 is the encryption of m and c_2 is the encryption of $\mathsf{m} + 2^{10}$. Determine the message m.

These ideas have been extended by Coppersmith, Franklin, Patarin and Reiter [144]. Among other things they study how to break related encryptions for any polynomial relation by using resultants (see Exercise 24.4.8).

Exercise 24.4.8. Let (N, e) be an RSA key. Suppose one is given $\mathsf{c}_1 = \mathsf{m}_1^e \pmod N$, $\mathsf{c}_2 = \mathsf{m}_2^e \pmod N$ and a polynomial $P(x, y) \in \mathbb{Z}[x, y]$ such that $P(\mathsf{m}_1, \mathsf{m}_2) \equiv 0 \pmod N$. Let d be a bound on the total degree of $P(x, y)$. Show how to compute m_1 and m_2 in $O((d+e)^3 d^2 M(\log(N)))$ bit operations.

24.4.5 Fixed Pattern RSA Signature Forgery

The aim of this section is to present a simple padding scheme, often called **fixed pattern padding** for RSA. We then sketch why this approach may not be sufficient to obtain RSA signatures secure against adaptive attackers. These ideas originate in the work

[9] The idea was presented at the "rump session" of CRYPTO 1995.

of De Jonge and Chaum and later work by Girault and Misarsky. We present the more recent attacks by Brier, Clavier, Coron and Naccache [106]. An attack on RSA encryption with fixed padding is given in Section 19.4.1.

Example 24.4.9. Suppose we are using moduli of length 3072 bits and that messages (or message digests) m are of length at most 1000 bits.

The padding scheme uses a fixed value $P = 2^{3071}$ and the signature on the message digest m (such that $0 \leq \mathsf{m} < 2^{1000}$) is

$$\mathsf{s} = (P + \mathsf{m})^d \pmod{N}.$$

The verifier computes $\mathsf{s}^e \pmod{N}$ and checks it is of the correct form $P + \mathsf{m}$ with $0 \leq \mathsf{m} < 2^{1000}$.

The following method (from [106]) forges signatures if messages are roughly $N^{1/3}$ in size. We assume that a signing oracle is available (we assume the signing oracle will only generate signatures if the input is correctly padded) and that a hash function is not applied to the messages. Suppose m is the target message, so we want to compute the d-th power of $z = P + \mathsf{m}$. The idea is to find small values u, v, w such that

$$z(z + u) \equiv (z + v)(z + w) \pmod{N}. \tag{24.2}$$

Then given signatures on $\mathsf{m} + u, \mathsf{m} + v$ and $\mathsf{m} + w$ (i.e., d-th powers of $z + u, z + v$ and $z + w$) one can compute the signature on m as required.

To find small solutions to equation (24.2) we expand and simplify to

$$z(u - v - w) \equiv vw \pmod{N}.$$

Running the extended Euclidean algorithm (the basic version rather than the fast version of Algorithm 1) on z and N gives a number of integers s, r such that

$$zs \equiv r \pmod{N} \quad \text{and} \quad |rs| \approx N.$$

One can run Euclid until a solution with $|s| \approx N^{1/3}$ and $|r| \approx N^{2/3}$ is found. One then tries to factor r as a product $r = vw$ of numbers of a similar size. If this is feasible (for example, if r has a large number of small prime factors) then set $u = s + v + w$ and we have a solution. This approach is reasonable as long as the messages are at least one third of the bit-length of the modulus.

Example 24.4.10. Let $N = 1043957 \approx 2^{20}$.

Suppose $P = 2^{19}$ is the fixed padding and suppose messages are restricted to be 10-bit binary strings. Thus

$$z = P + \mathsf{m}$$

where $0 \leq \mathsf{m} < 2^{10}$.

Suppose have access to a signing oracle and would like to forge a signature on the message $\mathsf{m} = 503$ that corresponds to $z = P + 503 = 524791$.

We apply Euclid's algorithm on N and z to solve the congruence $zs \equiv r \pmod{N}$ where $s \approx N^{1/3} \approx 101$.

i	q	r_i	s_i	t_i
-1	–	1043957	1	0
0	–	524791	0	1
1	1	519166	1	-1
2	1	5625	-1	2
3	92	1666	93	-185

This gives the solution $-185z \equiv 1666 \pmod{N}$ and $|-185| \approx N^{1/3}$. So set $s = -185$ and $r = 1666$. We try to factor r and are lucky that $1666 = 2 \cdot 7^2 \cdot 17$. So choose $v = 34$ and $w = 49$. Finally, choose $u = v + w - 185 = -102$. One can check that

$$z(z+u) \equiv (z+v)(z+w) \pmod{N}$$

and that $z+u, z+v$ and $z+w$ are all between P and $P + 2^{10}$. Hence, if one obtains signatures $\mathsf{s}_1, \mathsf{s}_2, \mathsf{s}_3$ on $\mathsf{m}+u = 401, \mathsf{m}+v = 537$ and $\mathsf{m}+w = 552$ then one has the signature on z as $\mathsf{s}_2\mathsf{s}_3\mathsf{s}_1^{-1} \pmod{N}$.

The success of this attack depends on the cost of factoring r and the probability that it can be written as a product of integers of similar size. Hence, the attack has subexponential complexity. For fixed m the attack may not succeed (since r might not factor as a product of integers of the required size). On the other hand, if m can vary a little (this is now more like an existential forgery) then the attack should succeed. A method for existential forgery that does not require factoring is given in Example 24.4.13.

Exercise 24.4.11. Give a variant of the above attack for the case where messages can be of size $N^{1/2}$ and for which it is only necessary to obtain signatures on two messages.

Exercise 24.4.12. One could consider affine padding $A\mathsf{m} + B$ instead of $P + \mathsf{m}$, where A and B are fixed integers and m is small. Show that, from the point of view of attacks, the two padding schemes are equivalent.

Example 24.4.13. We show sketch the existential forgery from [106]. As before we seek messages $\mathsf{m}_1, \ldots, \mathsf{m}_4$ of size $N^{1/3}$ such that

$$(P+\mathsf{m}_1)(P+\mathsf{m}_2) \equiv (P+\mathsf{m}_3)(P+\mathsf{m}_4) \pmod{N}.$$

Writing $\mathsf{m}_1 = x+t, \mathsf{m}_2 = y+t, \mathsf{m}_3 = t$ and $\mathsf{m}_4 = x+y+z+t$ the equation is seen to be equivalent to $Pz \equiv xy - tz \pmod{N}$. One again uses Euclid to find $s \approx N^{1/3}, r \approx N^{2/3}$ such that $Ps \equiv r \pmod{N}$. One sets $z = s$ and then wants to find x, y, t such that $xy = r + tz$. To do this choose a random integer $N^{1/3} < y < 2N^{1/3}$ such that $\gcd(y, z) = 1$ and set $t \equiv -z^{-1}r \pmod{y}$. One then easily solves for the remaining values and one can check that the m_i are roughly of the right size.

For further details and results we refer to Brier, Clavier, Coron and Naccache [106] and Lenstra and Shparlinski [374].

This idea, together with other techniques, has been used to cryptanalyse the ISO/IEC 9796-1 signature standard with great success. We refer to Coppersmith, Coron, Grieu, Halevi, Jutla, Naccache and Stern [143].

24.4.6 Two Attacks by Bleichenbacher

Example 24.4.14. (Bleichenbacher) Consider a padding scheme for RSA signatures with $e = 3$ that is of the following form.

00 01	FF FF $\cdots$ FF	Special block	$H(\mathsf{m})$

In other words, to verify a signature s one computes $\mathsf{s}^3 \pmod{N}$ and checks if the resulting integer corresponds to a binary string of the above form.

Suppose now that the verification algorithm parses the binary string from the left hand side (most significant bit) and does not check that $H(\mathsf{m})$ sits in the least significant bits (this was the case for some padding schemes in practice). In other words, a signature will verify if $\mathsf{s}^3 \pmod{N}$ is an integer whose binary representation is as follows, where r is any binary string.

00 01	FF FF $\cdots$ FF	Special block	$H(\mathsf{m})$	r

Bleichenbacher noticed that a forger could choose r to ensure that the integer is a cube in $\mathbb{Z}$.

Precisely, suppose $2^{3071} < N < 2^{3072}$, that the "special block" is 0000 (i.e., 32 zero bits), and that H has 256-bit output. Let m be a message such that $H(\mathsf{m}) \equiv 1 \pmod 3$. We want to find r such that

$$y = 2^{3057} - 2^{2360} + H(\mathsf{m})2^{2072} + r$$

is a cube. Note that

$$(2^{1019} - (2^{288} - H(\mathsf{m}))2^{34}/3)^3 = 2^{3057} - 2^{2360} + H(\mathsf{m})2^{2072} + 2^{1087}((2^{288} - H(\mathsf{m}))/3)^2 + z$$

where $|z| < 2^{980}$ and so is of the right form. To find the right value one can take an integer of the form y, take its cube root in $\mathbb{R}$ and then round up to the nearest integer.

Exercise 24.4.15. Compute a signature using the method of example 24.4.14 for the hash value $H(\mathsf{m}) = 4$. Check your answer.

Bleichenbacher has also given a chosen ciphertext attack on RSA encryption when using a fixed padding scheme [67]. More precisely, suppose a message m is padded as in the figure below to form an integer x, and is then encrypted as $\mathsf{c} = x^e \pmod N$.

00 02	non-zero padding string	00	m

Bleichenbacher supposes an attacker has access to an oracle that determines, given an integer c, whether the corresponding e-th root x has binary expansion in this form. Such an oracle is provided by a decryption oracle that either outputs m or $\perp$. Error messages from a server may also provide such an oracle.

We do not have space to give the details. The basic idea is that, given a challenge c, one computes $\mathsf{c}' = \mathsf{c}r^e \pmod N$ for various integers r and determines intervals containing the message according to the error response. The attack eventually reveals the message (after perhaps a million queries to the oracle).

24.5 Attacks on RSA Parameters

In this section we briefly recall some attacks on certain choices of RSA public key.

24.5.1 Wiener Attack on Small Private Exponent RSA

One proposal to speed-up RSA decryption is to choose d to be a small integer. Key generation is performed by first choosing d and then setting $e = d^{-1} \pmod{\lambda(N)}$. This is called **small private exponent RSA**.[10] We present the famous **Wiener attack**, which is a polynomial-time attack on private exponents $d < N^{1/4}$.

Exercise 24.5.1. Give a brute-force attack on small private exponent RSA that tries each odd integer $d > 1$ in turn. What is the complexity of this attack?

[10]The reader should remember that, in practice, it is more efficient to use the Chinese remainder theorem to speed up RSA decryption than small private exponents.

We now sketch Wiener's idea [630]. We assume the key generation of Figure 24.1 is used, so that $N = pq$ where $p < q < 2p$. Consider the equation defining e and d

$$ed = 1 + k\varphi(N)$$

(a similar attack can be mounted using the equation $ed = 1 + k\lambda(N)$, see Exercise 24.5.5). Since $e < \varphi(N)$ we have $k < d$. Now $\varphi(N) = N + 1 - (p + q)$ and $\sqrt{N} \leq (p + q) < 3\sqrt{N}$ so $\varphi(N) = N - u$ where $0 \leq u = p + q - 1 \leq 3\sqrt{N}$. Rearranging gives

$$-ed + kN = (-1 + ku) < 3k\sqrt{N}. \tag{24.3}$$

If d is smaller than $\sqrt{N}/3$ then the right hand side is $< N$. Hence, one could try to find d by running the extended Euclidean algorithm on (e, N) and testing the coefficient of e to see if it is a candidate value for $\pm d$ (e.g., by testing whether $(x^e)^d \equiv x \pmod{N}$ for a random $1 < x < N$). Note that one must use the basic extended Euclidean algorithm rather than the faster variant of Algorithm 1. We now explain that this method is guaranteed to find d when it is sufficiently small.

Theorem 24.5.2. *Let $N = pq$ where $p < q < 2p$ are primes. Let $e = d^{-1} \pmod{\varphi(N)}$ where $0 < d < N^{1/4}/\sqrt{3}$. Then given (N, e) one can compute d in polynomial time.*

Proof: Using the notation above, $(d, k, uk - 1)$ is a solution to equation (24.3) with $0 < k < d$ and $0 \leq u < 3\sqrt{N}$.

The Euclidean algorithm finds all triples (s, t, r) satisfying $es + Nt = r$ with $|sr| < N$ and $|tr| < e$. Hence, if $|d(uk - 1)| < N$ then the required solution will be found. If $0 < d < N^{1/4}/\sqrt{3}$ then

$$|duk| < d^2 u < \frac{N^{1/2}}{3} 3\sqrt{N} = N$$

which completes the proof. □

Example 24.5.3. Let $N = 86063528783122081$ with $d = 8209$. One computes that $e = 14772019882186053$.

One can check that

$$ed = 1 + 1409\varphi(N).$$

Running Euclid's algorithm with $r_{-1} = N$ and $r_0 = e$ and writing s_i, t_i be such that $r_i = s_i N + t_i e$ one finds the following table of values.

i	q	r_i	s_i	t_i
-1	$--$	86063528783122081	1	0
0	$--$	14772019882186053	0	1
1	5	12203429372191816	1	-5
2	1	2568590509994237	-1	6
3	4	1929067332214868	5	-29
4	1	639523177779369	-6	35
5	3	10497798876761	23	-134
6	60	9655245173709	-138	8075
7	1	842553703052	1409	-8209

One sees that d is found in only 7 steps.

Exercise 24.5.4. Consider the RSA public key $(N, e) = (11068562742977, 10543583750987)$. Use the Wiener attack to determine the private key.

Exercise 24.5.5. Show how to perform the Wiener attack when $\varphi(N)$ is replaced by $\lambda(N)$. What is the bound on the size of d for which the attack works?

Exercise 24.5.6. Let $(N, e) = (63875799947551, 4543741325953)$ be an RSA public key where $N = pq$ with $\gcd(p-1, q-1) > 2$ and small private exponent d such that $ed \equiv 1 \pmod{\lambda(N)}$. Use the Wiener attack to find d.

Exercise 24.5.7. Show that one can prevent the Wiener attack by adding a sufficiently large multiple of $\varphi(N)$ to e.

Wiener's result has been extended in several ways. Dujella [184] and Verheul and van Tilborg [621] show how to extend the range of d, while still using Euclid's algorithm. Their algorithms are exponential time. Boneh and Durfee [78] used lattices to extend the attack to $d < N^{0.284}$ and, with significant further work, extended the range to $d < N^{0.292}$. Blömer and May [71] give a simpler formulation of the Boneh-Durfee attack for $d < N^{0.284}$. Some unsuccessful attempts to extend Wiener's method to larger d are discussed by Suk [595] and Bauer [31].

24.5.2 Small CRT Private Exponents

As mentioned in Section 24.1.1, a common way to speed up RSA decryption is to use the Chinese remainder theorem. Indeed, one can choose the CRT private exponents d_p and d_q to be small (subject to $d_p \equiv d_q \pmod{\gcd(p-1, q-1)}$ and define e such that $ed_p \equiv 1 \pmod{p-1}$ and $ed_q \equiv 1 \pmod{q-1}$. Of course, one should take $d_p \neq d_q$, or else one can just apply the Wiener attack. We now show that these values cannot be taken to be too small.

Exercise 24.5.8. Give a brute-force attack on small private CRT exponents.

We now present a "birthday attack", which is attributed to Pinch in [492]. Let d_p be such that $ed_p \equiv 1 \pmod{p-1}$ and $ed_p \not\equiv 1 \pmod{q-1}$. Suppose we know that $1 < d_p < K$ and let $L = \lceil \sqrt{K} \rceil$. Then $d_p = d_0 + Ld_1$ where $0 \leq d_0, d_1 < L$ and, for a random integer $1 < \mathsf{m} < N$ one expects

$$\gcd(\mathsf{m}^{ed_0 - 1}\mathsf{m}^{Led_1} - 1, N) = p.$$

The problem is to detect this match. The idea is to use the method of Section 2.16 for evaluating polynomials. So, define

$$G(x) = \prod_{j=0}^{L-1} (\mathsf{m}^{ej-1}x - 1) \pmod{N}.$$

This polynomial has degree L and can be constructed using the method in the proof of Theorem 2.16.1 in $O(M(L)\log(L)M(\log(N)))$ bit operations. The polynomial $G(x)$ requires $L\log_2(N)$ bits of storage.

Now, compute $\mathsf{c} = \mathsf{m}^{Le} \pmod{N}$. We wish to evaluate $G(\mathsf{c}^{d_1}) \pmod{N}$ for each of the candidate values $0 \leq d_1 < L$ (to obtain a list of L values). This can be performed using Theorem 2.16.1 in $O(L\log(L)^2\log(\log(L))M(\log(N)))$ bit operations. For each value $G(\mathsf{c}^{d_1}) \pmod{N}$ in the list we can compute

$$\gcd(G(\mathsf{c}^{d_1}), N)$$

to see if we have split N. The total running time of the attack is $\tilde{O}(\sqrt{K})$ bit operations.

Exercise 24.5.9. (Galbraith, Heneghan and McKee [220]) Suppose one chooses private CRT exponents of bit-length n and Hamming weight w. Use the ideas of this section together with those of Section 13.6 to give an algorithm to compute a CRT private exponent, given n and w, with complexity $O(\sqrt{w}W\log(W)^2\log(\log(W))M(\log(N)))$ bit operations where $W = \binom{n/2}{w/2}$.

When e is also small (e.g., when using the key generation method of Exercise 24.1.5) then there are lattice attacks on small CRT private exponents. We refer to Bleichenbacher and May [69] for details.

24.5.3 Large Common Factor of $p-1$ and $q-1$

Variants of RSA have been proposed for moduli $N = pq$ where there is some integer r greater than 2 such that both $r \mid (p-1)$ and $r \mid (q-1)$. For example, as follows from the solution to Exercise 24.5.5, one can prevent the Wiener attack by taking r large. We explain in this section why such variants of RSA must be used with caution.

First we remark, following McKee and Pinch [414], that r should not be considered as a secret. This is because r is a factor of $N-1$ and so the elliptic curve method or the Pollard rho factoring method can be used to compute a, usually short, list of possible values for r. Note that there is no way to determine the correct value of r from the list, but the attacks mentioned below can be repeated for each candidate value for r. Certainly, if r is small then it can be easily found this way. Even if r is large, since factoring $N-1$ in this setting is not harder than factoring N, it follows that the problem of computing r is not harder than the most basic assumption underlying the scheme.

Even if r is not known, as noted by McKee and Pinch [414], it cannot be too large: applying the Pollard rho method by iterating the function

$$x \mapsto x^{N-1} + 1 \pmod{N}$$

will produce a sequence that repeats modulo p after $O(\sqrt{p/r})$ terms, on average. Hence, if r is too large then the factorisation of N will be found even without knowing r.

We now explain a method to factor N when r is known. Suppose $N = pq$ where $p < q < 2p$ are primes. Write

$$p = xr + 1\,, \quad q = yr + 1.$$

Then

$$(N-1)/r = xyr + (x+y) = ur + v \tag{24.4}$$

where u and v $(0 \leq v < r)$ are known and x, y are unknown.

Exercise 24.5.10. Let the notation be as above. Show that if $r > \sqrt{3}N^{1/4}$ then one can determine x and y in polynomial-time.

Exercise 24.5.11. (McKee and Pinch [414]) Let the notation be as above and suppose that $r < \sqrt{3}N^{1/4}$. Write

$$x + y = v + cr\,, \quad xy = u - c$$

where $c \in \mathbb{N}$. Show that $c < 3N^{1/2}/r^2$. Then show that

$$(x-y)^2 = r^2c^2 + (2rv+4)c + v^2 - 4u.$$

Hence, show how to determine c by exhaustive search in $O(N^{1/2}\log(N)^2/r^2)$ bit operations.

Exercise 24.5.12. Let the notation be as above. Show that the exponent of $(\mathbb{Z}/N\mathbb{Z})^*$ divides xyr. Hence, deduce that

$$z^{ur} \equiv z^{xyr+cr} \equiv z^{cr} \pmod{N}$$

for every $z \in (\mathbb{Z}/N\mathbb{Z})^*$. Given r, show how to find c (and hence split N) in an expected $O(N^{1/4}M(\log(N))/r)$ bit operations using the Pollard kangaroo algorithm (one could also use baby-step-giant-step).

Exercise 24.5.13. Suppose $N = pq$ where p and q are 1536-bit primes such that $p-1$ and $q-1$ have a large common factor r. Show that, to ensure security against an attacker who can perform 2^{128} operations, one should impose the restriction $1 \leq r < 2^{640}$.

Exercise 24.5.14. Generalise the above attacks to the case where $r \mid (p+1)$ and $r \mid (q+1)$.

24.6 Digital Signatures Based on RSA and Rabin

There are numerous signature schemes based on RSA and Rabin. Due to lack of space we just sketch two schemes in the random oracle model. Hohenberger and Waters [292] have given an RSA signature scheme in the standard model whose security relies only on the Strong-RSA assumption.

24.6.1 Full Domain Hash

A simple way to design RSA signatures that are secure in the random oracle model is to assume each user has a hash function $H : \{0,1\}^* \to (\mathbb{Z}/N\mathbb{Z})^*$ where N is their public key.[11] Such a hash function is called a **full domain hash**, since the hash output is the entire domain of the RSA trapdoor permutation. Constructing such a hash function is not completely trivial; we refer to Section 3.6. The signature on a message m in this case is $\mathsf{s} = H(\mathsf{m})^d \pmod{N}$. These ideas were formalised by Bellare and Rogaway, but we present the slightly improved security result due to Coron.

Theorem 24.6.1. *RSA signatures with full domain hash (**FDH-RSA**) have UF-CMA security in the random oracle model (i.e., where the full domain hash function is replaced by a random oracle) if the RSA problem is hard.*

Proof: (Sketch) Let A be an perfect[12] adversary playing the UF-CMA game. We build a simulator that takes an instance (N, e, y) of the RSA problem and, using A as a subroutine, tries to solve the RSA problem.

The simulator in this case starts by running the adversary A on input (N, e). The adversary will make queries to the hash function H, and will make decryption queries. The adversary will eventually output a pair $(\mathsf{m}^*, \mathsf{s}^*)$ such that s^* is a valid signature on m^*. To explain the basic idea of the simulator we remark that if one could arrange that $H(\mathsf{m}^*) = y$ then $(\mathsf{s}^*)^e \equiv y \pmod{N}$ and the RSA instance is solved.

The simulator simulates the random oracle H in the following way. First, the simulator will maintain a list of pairs $(\mathsf{m}, H(\mathsf{m}))$ where m was a query to the random oracle and $H(\mathsf{m}) \in (\mathbb{Z}/N\mathbb{Z})^*$ was the value returned. This list is initially empty. For each query m to the random oracle the simulator first checks if m has already been queried and, if

[11] In practice one designs $H : \{0,1\}^* \to \{0, 1, \ldots, N-1\}$ since the probability that a random element of $\mathbb{Z}/N\mathbb{Z}$ does not lie in $(\mathbb{Z}/N\mathbb{Z})^*$ is negligible.

[12] The proof in the general case, where the adversary succeeds with non-negligible probability ϵ, requires minor modifications.

so, responds with the same value $H(\mathsf{m})$. If not, the simulator chooses a random element $1 < r < N$, computes $\gcd(r, N)$ (and if this is not 1 then factors N, solves the RSA instance, and halts), computes $z = r^e \pmod N$ and with some probability $1 - p$ (we determine p at the end of the proof) returns z as $H(\mathsf{m})$ and with probability p returns $yz \pmod N$. The information $(\mathsf{m}, H(\mathsf{m}), r)$ is stored.

When the simulator is asked by the adversary to sign a message m it performs the following: First it computes $H(\mathsf{m})$ and the corresponding value r. If $H(\mathsf{m}) = r^e \pmod N$ then the simulator returns $\mathsf{s} = r$. If $H(\mathsf{m}) = yr^e \pmod N$ then the simulator fails.

Eventually, the adversary outputs a pair $(\mathsf{m}^*, \mathsf{s}^*)$. If $H(\mathsf{m}^*) = yr^e \pmod N$ where r is known to the simulator, and if $(\mathsf{s}^*)^e \equiv H(\mathsf{m}^*) \pmod N$, then $y = (\mathsf{s}^* r^{-1})^e \pmod N$ and so the simulator returns $\mathsf{s}^* r^{-1} \pmod N$. Otherwise, the simulator fails.

To complete the proof it is necessary to argue that the simulator succeeds with non-negligible probability. If the adversary makes q_S signing queries then the probability that the simulator can answer all of them is $(1-p)^{q_S}$. The probability that the message m^* corresponds to a random oracle query that allows us to solve the RSA problem is p. Hence, the probability of success is (ignoring some other negligible factors) $(1-p)^{q_S} p$. Assume that $q_S \geq 1$ and that q_S is known (in practice, one can easily learn a rough estimate of q_S by experimenting with the adversary A). Choose $p = 1/q_S$ so that the probability of success is $(1 - 1/q_S)^{q_S} \frac{1}{q_S}$, which tends to $1/(eq_S)$ for large q_S (where $e = 2.71828\ldots$). Since a polynomial-time adversary can only make polynomially many signature queries the result follows. We refer to Coron [148] for all the details. □

One problem with the full domain hash RSA scheme is the major loss of security (by a factor of q_S) in Theorem 24.6.1. In other words, the reduction is not tight. This can be avoided by including an extra random input to the hash function. In other words, an RSA signature is $(\mathsf{s}_1, \mathsf{s}_2)$ such that $\mathsf{s}_2^e \equiv H(\mathsf{m} \| \mathsf{s}_1) \pmod N$. Then, when the simulator is asked to output a signature on message m, it can choose a "fresh" value s_1^* and define $H(\mathsf{m} \| \mathsf{s}_1^*) = (\mathsf{s}_2^*)^e \pmod N$ as above. This approach avoids previous queries to $H(\mathsf{m} \| \mathsf{s}_1)$ with high probability. Hence, the simulator can answer "standard" hash queries with $yr^e \pmod N$ and "special" hash queries during signature generation with $r^e \pmod N$. This scheme is "folklore", but the details are given in Appendix A of Coron [149]. The drawback is that the extra random value s_1 must be included as part of the signature. The **PSS** signature padding scheme was designed by Bellare and Rogaway [41] precisely to allow extra randomness in this way without increasing the size of the signature. We refer to [149] for a detailed analysis of RSA signatures using the PSS padding.

Exercise 24.6.2. Give a security proof for the RSA full domain hash signature scheme with verification equation $H(\mathsf{m} \| \mathsf{s}_1) = \mathsf{s}_2^e \pmod N$.

The above results are all proved in the random oracle model. Paillier [475] has given some evidence that full domain hash RSA and RSA using PSS padding cannot be proved secure in the standard model. Theorem 1 of [475] states that if one has a "black box" reduction from the RSA problem to selective forgery for a signature scheme under a passive attack, then under an adaptive chosen message attack one can, in polynomial-time, forge any signature for any message.

24.6.2 Secure Rabin-Williams Signatures in the Random Oracle Model

In this section we give a tight security result, due to Bernstein [47], for Rabin signatures. We assume throughout that $N = pq$ is a **Williams integer**; in other words, a product of primes $p \equiv 3 \pmod 8$ and $q \equiv 7 \pmod 8$ (such integers were discussed in Section 24.2.1).

We assume that $H : \{0,1\}^* \to \{0,1\}^\kappa$ is a cryptographic hash function (which will be modelled as a random oracle) where $2^\kappa < N < 2^{\kappa+O(\log(\kappa))}$.

Exercise 24.6.3. Suppose $p \equiv 3 \pmod 8$ and $q \equiv 7 \pmod 8$ are primes and $N = pq$. Then $(\frac{-1}{p}) = (\frac{-1}{q}) = (\frac{2}{p}) = -1$ while $(\frac{2}{q}) = 1$. Show that, for any integer $h \in (\mathbb{Z}/N\mathbb{Z})^*$, there are unique integers $e \in \{1, -1\}$ and $f \in \{1, 2\}$ such that efh is a square modulo N.

The signature scheme for public key N is as follows. For a message $\mathsf{m} \in \{0,1\}^*$ one computes $H(\mathsf{m})$ and interprets it as an integer modulo N (with overwhelming probability, $H(\mathsf{m}) \in (\mathbb{Z}/N\mathbb{Z})^*$). The signer determines the values e and f as in Exercise 24.6.3 and determines the four square roots s_1, s_2, s_3, s_4 satisfying $s_i^2 \equiv H(\mathsf{m})ef \pmod N$. The signer then deterministically chooses one of the values s_i (for example, by ordering the roots as integers $s_1 < s_2 < s_3 < s_4$ and then generating an integer $i \in \{1, 2, 3, 4\}$ using a pseudorandom number generator with a secret key on input m). The signature is the triple $\mathsf{s} = (e, f, (ef)^{-1}s_i \pmod N)$. It is crucially important that, if one signs the same message twice, then the same signature is output. To verify a signature $\mathsf{s} = (e, f, s)$ for public key N and message m one computes $H(\mathsf{m})$ and then checks that $efs^2 \equiv H(\mathsf{m}) \pmod N$.

Exercise 24.6.4. Show that if a signer outputs two signatures (e_1, f_1, s_1) and (e_2, f_2, s_2) for the same message m such that $s_1 \not\equiv \pm s_2 \pmod N$ then one can factor the modulus.

Exercise 24.6.5. Show that it is not necessary to compute the Jacobi symbol $(\frac{H(\mathsf{m})}{N})$ when generating Rabin-Williams signatures as above. Instead, one can compute $H(\mathsf{m})^{(p+1)/4} \pmod p$ and $H(\mathsf{m})^{(q+1)/4} \pmod q$, as is needed to compute the s_i, and determine e and f with only a little additional computation.

Theorem 24.6.6. *The Rabin-Williams signature scheme sketched above has UF-CMA security in the random oracle model (i.e., if H is replaced by a random oracle) if factoring Williams integers is hard and if the pseudorandom generator is indistinguishable from a random function.*

Proof: (Sketch) Let A be a perfect adversary against the Rabin-Williams signature scheme and let N be a Williams integer to be factored. The simulator runs the adversary A on N.

The simulator must handle the queries made by A to the random oracle. To do this it maintains a list of hash values, which is initially empty. When A queries H on m the simulator first checks whether m appears on the list of hash values, and, if it does, responds with the same value as previously. If H has not been previously queried on m the simulator chooses random $s \in (\mathbb{Z}/N\mathbb{Z})^*$, $e \in \{-1, 1\}$, $f \in \{1, 2\}$ and computes $h = efs^2 \pmod N$. If $0 \leq h < 2^\kappa$ then return h and store (m, e, f, s, h) in the list. If h is too big then repeat with a different choice for (s, e, f). Since $N < 2^{\kappa+O(\log(\kappa))}$ the expected number of trials is polynomial in $\log(N)$.

When A makes a signature query on m the simulator first queries $H(\mathsf{m})$ and gets the values (e, f, s) from the hash list such that $H(\mathsf{m}) \equiv efs^2 \pmod N$. The simulator can therefore answer with (e, f, s), which is a valid signature. (It is necessary to show that the values (e, f, s) output in this way are indistinguishable from the values output in the real cryptosystem, and this requires that the pseudorandom choice of s from among the four possible roots be computationally indistinguishable from random; note that the adversary cannot detect whether or not a pseudorandom generator has actually been used since it does not know the secret key for the generator.)

Finally, A outputs a signature (e^*, f^*, s^*) on a message m^*. Recalling the values (e, f, s) from the construction of $H(\mathsf{m}^*)$ we have $e^* = e$, $f^* = f$ and $(s^*)^2 \equiv s^2 \pmod N$.

With probability 1/2 we can factor N as $\gcd(N, s^* - s)$. We refer to Section 6 of Bernstein [47] for the full details. □

Exercise 24.6.7. Show that if one can find a collision for the hash function H in Bernstein's variant of Rabin-Williams signatures and one has access to a signing oracle then one can factor N with probability 1/2.

Exercise 24.6.8. Suppose the pseudorandom function used to select the square root in Rabin-Williams signatures is a function of $H(\mathsf{m})$ rather than m. Show that, in contrast to Exercise 24.6.7, finding a collision in H no longer leads to an algorithm to split N. On the other hand, show that if one can compute a preimage for H and one has access to a signing oracle then one can factor N with probability 1/2.

Exercise 24.6.9. Adapt the proof of Theorem 24.6.6 to the case where H has full domain output.

Exercise 24.6.10. Adapt the proof of Theorem 24.6.1 to the case where $H : \{0,1\}^* \to \{0,1\}^\kappa$ where $2^\kappa < N < 2^{\kappa + O(\log(\kappa))}$.

24.6.3 Other Signature and Identification Schemes

Example 24.6.11. Let (N, e) be an RSA public key for a trusted authority, where e is a large prime. Shamir [547] proposed the following identity-based signature scheme. A user with identity $\mathsf{id} \in \{0,1\}^*$ has a corresponding public key $H_1(\mathsf{id}) \in (\mathbb{Z}/N\mathbb{Z})^*$, where H_1 is a cryptographic hash function. The user obtains their private key s_{id} such that

$$s_{\mathsf{id}}^e \equiv H_1(\mathsf{id}) \pmod{N}$$

from the trusted authority.

To sign a message $\mathsf{m} \in \{0,1\}^*$ the user chooses a random integer $1 < r < N$, computes $\mathsf{s}_1 = r^e \pmod{N}$, then computes $\mathsf{s}_2 = s_{\mathsf{id}} r^{H_2(\mathsf{m} \| \mathsf{s}_1)} \pmod{N}$ where $H_2 : \{0,1\}^* \to \{0, 1, \ldots, N-1\}$ is a cryptographic hash function and s_1 is represented as a binary string. The signature is $(\mathsf{s}_1, \mathsf{s}_2)$.

To verify a signature one checks that

$$\mathsf{s}_2^e \equiv H_1(\mathsf{id}) \mathsf{s}_1^{H_2(\mathsf{m} \| \mathsf{s}_1)} \pmod{N}.$$

Exercise 24.6.12. Show that outputs $(\mathsf{s}_1, \mathsf{s}_2)$ of the Shamir signature algorithm do satisfy the verification equation.

The security of Shamir signatures was analysed by Bellare, Namprempre and Neven [35] (also see Section 4 of [335]). They show, in the random oracle model, that the scheme is secure against existential forgery if the RSA problem is hard.

Exercise 24.6.13. Show how to break the Shamir signature scheme under a known message attack if e is small (for example, $e = 3$).

Exercise 24.6.14. Consider the following modified version of the Shamir identity-based signature scheme: The verification equation for signature $(\mathsf{s}_1, \mathsf{s}_2)$ on message m and identity id is

$$\mathsf{s}_2^e \equiv H_1(\mathsf{id}) \mathsf{s}_1^{H_2(\mathsf{m})} \pmod{N}.$$

Show how the user with private key s_{id} can generate signatures. Give a selective forgery under a known message attack for this scheme.

Exercise 24.6.15. Consider the following modified version of the Shamir identity-based signature scheme: The verification equation for signature s on message m and identity id is

$$\mathsf{s}^e \equiv H_1(\mathsf{id})^{H_2(\mathsf{m})} \pmod{N}.$$

Show how the user with private key s_{id} can generate signatures. Show how to compute the private key for this scheme under a known message attack.

We now briefly present two interactive identification schemes that are convenient alternatives to RSA and Rabin signatures for constrained devices. The notion of an identification scheme was sketched in Section 22.1.1; recall that it is an interactive protocol between a prover and a verifier.

Example 24.6.16. (Feige, Fiat and Shamir [201]) A prover has public key $(N, v_1, \ldots, v_k)$, where $N = pq$ is an RSA modulus and $1 < v_j < N$ for $1 \leq j \leq k$. The private key is a list of integers $u_1, \ldots, u_k$ such that $v_j \equiv u_j^2 \pmod{N}$ for $1 \leq j \leq k$. The identification protocol is as follows: The prover chooses a random integer $1 < r < N$ and send $\mathsf{s}_1 = r^2 \pmod{N}$ to the verifier. The verifier sends a challenge $c = (c_1, \ldots, c_k) \in \{0,1\}^k$. The prover computes

$$\mathsf{s}_2 = r \prod_{j=1}^{k} u_j^{c_j} \pmod{N}$$

and sends it to the verifier. The verifier checks that

$$\mathsf{s}_2^2 \equiv \mathsf{s}_1 \prod_{j=1}^{k} v_j^{c_j} \pmod{N}.$$

One can try to impersonate a user by guessing the challenge c and defining s_1 accordingly; this succeeds with probability $1/2^k$. The protocol can be repeated a number of times if necessary. For example, one might choose $k = 20$ and repeat the protocol 4 times.

The point is that both signing and verification are only a small number of modular multiplications (in contrast to Rabin signatures, for which signing requires computing two large exponentiations). The security is based on factoring (see Exercise 24.6.17).

The public key can be shortened by choosing $v_1, \ldots, v_k$ to be the first k primes. Alternatively, the scheme can be made identity-based by defining $v_1, \ldots, v_k$ as a function of the identity of a user; the user can get the values u_i from the trusted authority who knows the factorisation of N. The identification scheme can be turned into a signature scheme (either public key or identity-based) by choosing the value c as the output of a hash function $H(\mathsf{m} \| \mathsf{s}_1)$; but then k should be taken to be very large. For further discussion of these schemes see Sections 10.4.2 and 11.4.1 of Menezes, van Oorschot and Vanstone [418].

Exercise 24.6.17. Let A be an algorithm that takes as input a public key for the Feige-Fiat-Shamir scheme, outputs a value s_1, receives two distinct challenges c_1 and c_2 and outputs values $\mathsf{s}_{2,1}$ and $\mathsf{s}_{2,2}$ satisfying the verification equation for c_1 and c_2 respectively. Show how to use A to compute square roots modulo N.

Example 24.6.18. (Guillou and Quisquater [271]) A prover has public key (N, e, u) where $N = pq$ is an RSA modulus, e is an RSA exponent (i.e., $\gcd(e, \varphi(N)) = 1$) and $1 < u < N$ is a randomly chosen integer. The private key is an integer s such that $s^e \equiv u \pmod{N}$. The identification protocol is as follows: The prover chooses a random integer $1 < r < N$ and sends $\mathsf{s}_1 = r^e \pmod{N}$ to the verifier. The verifier sends a

challenge $0 \leq c < 2^k$ to the prover, who replies with $\mathsf{s}_2 = rs^c \pmod N$. The verifier checks that

$$\mathsf{s}_2^e \equiv \mathsf{s}_1 u^c \pmod N.$$

When e and c are small then both the prover and verifier have easy computations (in contrast with RSA, where at least one party must perform a large exponentiation). Hence this scheme can be more suitable than RSA in constrained environments. The security depends on the RSA problem (see Exercise 24.6.19).

One can try to impersonate a user by guessing the challenge c and defining s_1 accordingly; this succeeds with probability $1/2^k$. The protocol can be repeated a number of times if necessary.

The scheme can be made identity-based by replacing u with $H_1(\mathsf{id})$. Each user can get their private key s_{id} such that $s_{\mathsf{id}}^e \equiv H_1(\mathsf{id}) \pmod N$ from the trusted authority. The scheme can be turned into a signature scheme (either public key or identity-based) by setting $c = H(\mathsf{m} \| \mathsf{s}_1)$; but then k should be sufficiently large. A variant with lower bandwidth is to send only some bits of s_1 and change the verification equation to checking that the appropriate bits of $\mathsf{s}_2^e u^{-c} \pmod N$ equal those of s_1. For further discussion of these schemes see Sections 10.4.3 and 11.4.2 of Menezes, van Oorschot and Vanstone [418].

Exercise 24.6.19. Let A be an algorithm that takes as input a public key for the Guillou-Quisquater scheme, outputs a value s_1, receives two distinct challenges c_1 and c_2 and outputs values $\mathsf{s}_{2,1}$ and $\mathsf{s}_{2,2}$ satisfying the verification equation for c_1 and c_2 respectively. Show how to use A to solve the RSA problem.

24.7 Public Key Encryption Based on RSA and Rabin

24.7.1 Padding Schemes for RSA and Rabin

To prevent algebraic attacks such as those mentioned in Sections 1.2 and 24.4 it is necessary to use randomised padding schemes for encryption with RSA. Three goals of padding schemes for RSA are listed below.

1. To introduce randomness into the message and expand short messages to full size.
2. To ensure that algebraic relationships among messages do not lead to algebraic relationships between the corresponding ciphertexts.
3. To ensure that random elements of $\mathbb{Z}_N^*$ do not correspond to valid ciphertexts. This means that access to a decryption oracle in CCA attacks is not useful.

Example 24.7.1. Consider the following naive padding scheme when using 3072-bit RSA moduli: suppose messages are restricted to be at most 256 bits, and suppose a random 2815-bit value is r appended to the message to bring it to full length (i.e., the value input to the RSA function is $\mathsf{m} + 2^{256}r$). This certainly adds randomness and destroys many algebraic relationships. However, there is an easy CCA attack on the OWE-security of RSA with this padding scheme.

To see this, suppose $2^{3071} < N < 2^{3072}$ and let c be the ciphertext under attack. With probability $1/2$ the most significant bit of r is zero and so $\mathsf{c}' = \mathsf{c}2^e \pmod N$ is a valid padding of $2\mathsf{m}$ (except that the most significant bit of m is lost). Hence, by decrypting c' one determines all but the most significant bit of m. Similarly, if the least significant bit of m is zero then one can make a decryption query on $\mathsf{c}' = 2^{-e}\mathsf{c} \pmod N$.

Exercise 24.7.2. Consider the situation of Example 24.7.1 again. Find a CCA attack on RSA with the padding scheme $1+2\mathsf{m}+2^{257}r+2^{3071}$ where $0 \leq \mathsf{m} < 2^{256}$ and $0 \leq r < 2^{2813}$.

24.7.2 OAEP

In this section we present the **OAEP** (Optimal Asymmetric Encryption Padding) scheme, which was developed by Bellare and Rogaway [40] (for more details see Section 5.9.2 of Stinson [592]). The word "optimal" refers to the length of the padded message compared with the length of the original message: the idea is that the additional bits in an OAEP padding of a message are as small as can be.

The padding scheme takes as input an n-bit message m and outputs a κ-bit binary string S that can then be encrypted (in the case of RSA one encrypts by interpreting S as an element of $\mathbb{Z}_N^*$ and raising to the power e modulo N). Similarly, to decrypt a ciphertext c corresponding to an RSA-OAEP message one computes $\mathsf{c}^d \pmod N$, interprets this number as a bitstring S, and then unpads to get m or $\perp$. Figure 24.3 describes the OAEP scheme in detail. As usual, $a\|b$ denotes the concatenation of two binary strings and $a \oplus b$ denotes the bitwise XOR of two binary strings of the same length.

System Parameters: Suppose we are using $(\kappa+1)$-bit RSA moduli (e.g., $\kappa = 3071$). Let κ_0 and κ_1 be chosen so that no adversary can perform 2^{κ_i} operations, e.g. $\kappa_0 = \kappa_1 = 128$. Set $n = \kappa - \kappa_0 - \kappa_1$. Messages are defined to be n-bit strings.
Let G be a cryptographic hash function mapping κ_0 bit strings to $n+\kappa_1$ bit strings and H be a cryptographic hash function from $n+\kappa_1$ bit strings to κ_0 bit strings.

Pad: Let m be an n-bit string.

1. Choose a random κ_0-bit string R.
2. Set $S_1 = (m\|0^{\kappa_1}) \oplus G(R)$, where 0^{κ_1} denotes a κ_1-bit string of zeroes.
3. Set $S_2 = R \oplus H(S_1)$.
4. Set $S = S_1\|S_2$.

Unpad: Given a κ-bit string S one calls the low κ_0 bits S_2 and the high $n+\kappa_1$ bits S_1.

1. Compute $R = S_2 \oplus H(S_1)$.
2. Compute $S_3 = S_1 \oplus G(R)$.
3. Check whether the low κ_1 bits of S_3 are all zero, if not output $\perp$ and halt.
4. Obtain m as the top n bits of S_3.

Figure 24.3: OAEP.

The RSA-OAEP scheme does achieve the three goals of padding schemes mentioned earlier. First, the padding scheme is randomised and has full length. Second, since S_1 and S_2 are both XORs with outputs of hash functions, the bitstring S "looks random" and algebraic relationships among messages do not lead to algebraic relationships among their paddings. Third, if you have a decryption oracle and you send it a random element of $\mathbb{Z}_N^*$ then it will output $\perp$ with high probability.

The security of RSA-OAEP has a complicated history. Bellare and Rogaway [40] gave an IND-CCA security result, assuming that the RSA problem is hard, in the random oracle model, but Shoup [555] found a flaw in their security proof (though not an attack on their scheme). Shoup also gave a variant of OAEP (called OAEP+) together with a security proof in the random oracle model. Fujisaki, Okamoto, Pointcheval and Stern [215] were able to give a proof of the security of RSA using OAEP in the random oracle model by exploiting the random self-reducibility of RSA. The reader is referred to these papers,

and the excellent survey by Gentry [252], for the details.

24.7.3 Rabin-SAEP

Boneh [74] has considered a padding scheme (which he calls **SAEP**) that is simpler than OAEP and is suitable for encrypting short messages with Rabin. Note that the restriction to short messages is not a serious problem in practice, since public key encryption is mainly used to transport symmetric keys; nevertheless, this restriction means that SAEP is not an "optimal" padding. The scheme can also be used for RSA with extremely short public exponents (such as $e = 3$). We sketch the proof of security of Rabin encryption using this padding, as it is relatively straightforward and it is also a nice application of some of the cryptanalysis techniques already seen in Section 24.4.

Let $\kappa \in \mathbb{N}$ be a security parameter (for example, $\kappa = 3070$). Suppose $N = pq$ is an RSA modulus such that $2^{\kappa+1} < N < 2^{\kappa+1} + 2^{\kappa}$. We will also assume that $p \equiv q \equiv 3 \pmod 4$. Suppose $0 < \kappa_0, \kappa_1 < \kappa_0 + \kappa_1 < \kappa$ (later we will insist that $\kappa_1 > (\kappa + 2)/2$ and $0 < n = \kappa - \kappa_0 - \kappa_1 < \kappa/4$). Let $H : \{0,1\}^{\kappa_1} \to \{0,1\}^{n+\kappa_0}$ be a cryptographic hash function. The SAEP padding of a message $\mathsf{m} \in \{0,1\}^n$ is as follows:

1. Set $S_0 = \mathsf{m} \| 0^{\kappa_0}$.
2. Choose a random element $R \in \{0,1\}^{\kappa_1}$.
3. Set $S_1 = S_0 \oplus H(R)$.
4. Output $S = S_1 \| R$.

To unpad an SAEP bitstring S one first writes S as $S_1 \| R$ where $R \in \{0,1\}^{\kappa_1}$, then computes $S_0 = S_1 \oplus H(R)$, then writes $S_0 = \mathsf{m} \| S_2$ where $S_2 \in \{0,1\}^{\kappa_0}$. If S_2 is not the all zero string then return $\perp$, else return m.

Rabin-SAEP encryption proceeds by taking a message $\mathsf{m} \in \{0,1\}^n$, padding it as above, interpreting the bitstring as an integer $0 \leq x < 2^{\kappa} < N/2$, and computing $\mathsf{c} = x^2 \pmod N$. As usual, with overwhelming probability we have $\gcd(x, N) = 1$, so we assume that this is the case.

To decrypt the ciphertext c one computes the four square roots $1 \leq x_1, \ldots, x_4 < N$ of c modulo N in the usual way. Writing these such that $x_3 = N - x_1, x_4 = N - x_2$ it follows that at exactly two of them are less than $N/2$. Hence, at most two are less than 2^{κ}. For each root x such that $x < 2^{\kappa}$ perform the unpad algorithm to get either $\perp$ or a message m. If there are two choices for x, and either both give $\perp$ or both give messages, then output $\perp$. Otherwise, output m. An integer c is said to be a **valid ciphertext** if it decrypts to a message m, otherwise it is an **invalid ciphertext**.

Exercise 24.7.3. Show that if S is the output of the pad algorithm on $\mathsf{m} \in \{0,1\}^n$ then m is the output of unpad on S. Then show that if c is a Rabin-SAEP encryption of a message $\mathsf{m} \in \{0,1\}^n$ then the decryption algorithm on c outputs m with overwhelming probability.

We will now study the security of the scheme. Intuitively, a CCA attacker of the Rabin-SAEP scheme either computes at least one square root of the challenge ciphertext c^*, or else computes a ciphertext c related to c^* in the sense that if c^* is an encryption of m then c is an encryption of $\mathsf{m} \oplus z$ for some bitstring z. In this latter case, there is a square root of c that differs from the desired square root of c^* only in some of the most significant n bits. The proof of Theorem 24.7.5 shows how either situation leads to the factorisation of N.

Lemma 24.7.4. *Let $\kappa \in \mathbb{N}$ (with $\kappa > 10$). Let $N = pq$, where $p \equiv q \equiv 3 \pmod 4$ are prime. Suppose further that $2^{\kappa+1} < N < 2^{\kappa+1} + 2^{\kappa}$. Let $y = x^2 \pmod N$ where $0 < x < 2^{\kappa}$ is randomly chosen. Then there is an integer $0 < x' < 2^{\kappa}$ such that $x' \neq x$ but $y \equiv (x')^2 \pmod N$ with probability at least $1/3$.*

Proof: We know that $2^{\kappa} < N/2 < \frac{3}{2}2^{\kappa}$.

Let $A = \{0 < x < 2^{\kappa} : \gcd(x, N) = 1$ and $x^2 \pmod N$ has exactly one square root in the range$\}$ and $B = \{0 < x < 2^{\kappa} : \gcd(x, N) = 1$ and $x^2 \pmod N$ has exactly two square roots in the range$\}$. Note that $\#A + \#B \geq 2^{\kappa} - p - q + 1$.

Since every such $y = x^2 \pmod N$ has exactly two square roots between 0 and $N/2$, it follows that for each $x \in A$ then the other square root x' satisfies $2^{\kappa} \leq x' < N/2$. Hence, $\#A < N/2 - 2^{\kappa}$ and so $\#B > 2^{\kappa+1} - N/2 - p - q$.

Finally, the probability of having two roots is

$$\frac{\#B}{2^{\kappa}} > 2 - \frac{N}{2^{\kappa+1}} - \frac{p+q}{2^{\kappa}} > \frac{1}{2} - \frac{p+q}{2^{\kappa}},$$

which is $1/2 - \epsilon$ for some small $\epsilon > 0$, and certainly greater than $1/3$ when $p, q > 2^5$. □

Theorem 24.7.5. *Let $N = pq$ be a public key for the Rabin-SAEP encryption scheme. Suppose $0 < \kappa_0, \kappa_1 < \kappa$ are parameters such that $\kappa_1 > (\kappa+2)/2$ and $n = \kappa - \kappa_0 - \kappa_1 < \kappa/4$. If factoring Blum integers is hard then Rabin-SAEP has IND-CCA security in the random oracle model (i.e., when the hash function H is replaced by oracle access to a random function).*

Proof: (Sketch) Let A be an adversary against IND-CCA security of Rabin-SAEP in the random oracle model. We build a simulator that takes as input an integer $N = pq$ and attempts to factor it. The simulator will run A on the public key N.

The adversary A makes queries to the hash function oracle, decryption queries, and, at some point, gives two messages m_0 and m_1 and requests a challenge ciphertext c^* that is an encryption of m_b where $b \in \{0, 1\}$. The adversary eventually outputs a guess for the bit b. The simulator has to respond to these queries.

To respond to the request for the challenge ciphertext the simulator will choose a random $0 < a^{\dagger} < 2^{\kappa}$ and compute $\mathsf{c}^* = (a^{\dagger})^2 \pmod N$. By Lemma 24.7.4, with probability at least $1/3$ there is a second value $0 < a^* < 2^{\kappa}$ such that $\mathsf{c}^* \equiv (a^*)^2 \pmod N$. The entire security proof is designed so that the simulator has a chance to learn the value a^*. Once the simulator knows a^* it can factor N by computing $\gcd(N, a^{\dagger} - a^*)$. Without loss of generality, $a^{\dagger}$ is chosen at the start of the simulation.

Let $r^{\dagger}$ and r^* be the κ_1 least significant bits of $a^{\dagger}$ and a^* respectively. Note that $r^{\dagger}$ is known to the simulator but r^* is not. The question of whether c^* is a valid ciphertext, and if so, whether it encrypts message m_0 or m_1, depends entirely on the values $H(r^{\dagger})$, $H(r^*)$, $a^{\dagger}$ and a^*. Surprisingly, it is not necessary for the simulator to ensure that c^* is a valid encryption of either m_0 or m_1. The crux of the proof is the observation that, in order to be able to answer the above questions, one has to perform some operations involving r^*. For the remainder of the proof we assume that the value $H(r^{\dagger})$ is such that $a^{\dagger}$ does not correspond to a correct SAEP padding of a message. Further, it may help the reader to imagine that the value of H on r^* is such that $x^* \oplus H(r^*)$ (where $a^* = x^* \| r^*$) has its κ_0 least significant bits equal to zero.[13]

Boneh shows that either A makes a query to the hash function H on r^*, or A makes a query to the decryption oracle with a ciphertext c that is an encryption using the value r^*. These observations are exploited as follows.

[13] As we will see, the point is that this computation need never be performed, since as soon as r^* is queried to H the simulator wins and the game is over.

- If c is a ciphertext such that a corresponding value r has been queried to H, then the polynomial $F(y) = (y2^{\kappa_1} + r)^2 - \mathsf{c}$ has a root modulo N with $0 \leq y < \sqrt{N}$, and this can be efficiently found using Coppersmith's method (see Theorem 19.1.9; it is easy to make $F(y)$ monic before applying the theorem). For each decryption query on c one therefore repeats the above process for all values r that have been queried to H (we may assume at this point that $r \neq r^*$). A solution to this equation gives one of the roots of the ciphertext. In the case where the adversary is asking the simulator to decrypt a ciphertext that it has previously encrypted, then the simulator will respond correctly. We stress that if none of the values r queried to H correspond to the value c then there is not likely to be a small solution to the equation and Coppersmith's method does not provide any useful output.
- Alternatively, suppose c is a valid ciphertext that shares the value r^* with c^*. Let x be the corresponding square root of c, so that $\mathsf{c} = x^2 \pmod N$ and the κ_1 least significant bits of x equal r^*. Even though $H(r^*)$ has not yet been queried (otherwise, we have already factored N) if c^* and c are valid ciphertexts then we do not just have $x \equiv a^* \equiv r^* \pmod{2^{\kappa_1}}$ but $x \equiv a^* \pmod{2^{\kappa_0+\kappa_1}}$.

  Let z be such that $x = a^* + 2^{\kappa_0+\kappa_1} z$. Then $F_1(y) = y^2 - \mathsf{c}^*$ and $F_2(y, z) = (y + 2^{\kappa_0+\kappa_1} z)^2 - \mathsf{c}$ have a root y modulo N in common. Taking the resultant of $F_1(y)$ and $F_2(y, z)$ with respect to y gives a degree 4 polynomial $F(z)$ with a root $0 < z < N^{1/4}$. In such a situation, z can be computed by Coppersmith's theorem (again, it is easy to make the polynomial monic). Once z is computed one solves for y using the polynomial gcd and hence obtains the desired root a^* of c^*. These calculations will fail (in the sense that Coppersmith's algorithm does not output a small root) if c does not have a root x such that $x \equiv a^* \pmod{2^{\kappa_0+\kappa_1}}$.

All the main ideas are now in place, so we can finally describe the simulator. First we explain how the simulator handles queries to the oracle H.

1. The simulator creates a table of values $(r, H(r))$, which is initially set to have the single entry $(r^\dagger, H(r^\dagger))$ where $H(r^\dagger)$ is chosen uniformly at random.
2. If A queries the oracle H on a value r such that r already appears in the table, then the simulator returns the corresponding value $H(r)$.
3. If A queries the oracle H on a new value r then the simulator runs Coppersmith's algorithm on $F(y) = (y2^{\kappa_1} + r)^2 - \mathsf{c}^*$ in the hope of finding a root of c^*. If this succeeds then the root must be a^* (since if $r = r^\dagger$ we do not execute this step) and N is factored.
4. If r is new and Coppersmith's method does not yield a root of c^* then choose uniformly at random a value $H(r)$, add $(r, H(r))$ to the table, and return $H(r)$.

Now we explain how to handle decryption queries.

1. If A asks to decrypt a ciphertext c then, for each value r in the table of hash queries, run Coppersmith's algorithm on $F(y) = (y2^{\kappa_1} + r)^2 - \mathsf{c}$. If a square root of c is computed then perform the SAEP unpad algorithm and output the message or $\perp$ accordingly. Note that this is not exactly the same as the decryption algorithm, since it would check all square roots of c in the range of interest.
2. If step 1 does not succeed then compute the polynomial $F(z)$ as the resultant of $F_1(y) = y^2 - \mathsf{c}^*$ and $F_2(y, z) = (y + 2^{\kappa_0+\kappa_1} z)^2 - \mathsf{c}$ with respect to y as discussed above. If Coppersmith's method succeeds for $F(z)$ then the simulator can compute a^* and factor N.

3. If neither of these steps succeed then output $\perp$.

The main claim is that if A has a non-negligible probability of success then one of the above approaches for finding a^* succeeds with non-negligible probability. For complete details, and in particular, a careful analysis of the probabilities, we refer to Boneh [74]. □

Exercise 24.7.6. Prove the analogue of Theorem 24.7.5 when using SAEP padding with RSA and encryption exponent $e = 3$. Give the restriction on the sizes of n, κ_0 and κ_1 in this case.

24.7.4 Further Topics

Hybrid Encryption

It is standard in cryptography that encryption is performed using a **hybrid** system. A typical scenario is to use public key cryptography to encrypt a random session key $K_1 \| K_2$. Then the document is encrypted using a symmetric encryption scheme such as AES with the key K_1. Finally a MAC (message authentication code) with key K_2 of the symmetric ciphertext is appended to ensure the integrity of the transmission.

Encryption Secure in the Standard Model

Cramer and Shoup [160] have given an encryption scheme, using the universal hash proof systems framework based on the Paillier cryptosystem. Their scheme (using $N = pq$ where p and q are safe primes) has IND-CCA security in the standard model if the composite residuosity problem is hard (see Definition 24.3.4). We do not have space to present this scheme.

Hofheinz and Kiltz [291] have shown that the Elgamal encryption scheme of Section 23.1, when implemented in a certain subgroup of $(\mathbb{Z}/N\mathbb{Z})^*$, has IND-CCA security in the random oracle model if factoring is hard, and in the standard model if a certain higher residuosity assumption holds.