Chapter 10

Wiener's Attack

Michael J. Wiener proved in 1989 that, if $d < \frac{1}{3}N^{0.25}$, then one can fully recover d given (N, e). Boneh, Durfee, and Frankel improved this bound in 1998 to $d < N^{0.292}$. For $d < N^{0.5}$, it is expected that d can be recovered as well given (N, e), but it is still an open problem how this can be achieved.

One can exploit the Chinese Remainder Theorem to compute RSA inverses separately modulo p and q. For computing $c^d \bmod N$, one does the following:

- Compute u and v such that $up + vq = 1$.
- Compute $m_p := c^{d_p} \bmod p$ and $m_q := c^{d_q} \bmod q$ where $d_p = d \bmod p - 1$ and $d_q = d \bmod q - 1$.
- Set $m := upm_q + vqm_p \bmod N$.

The advantage is that all computations are carried out in significantly smaller groups, using exponents of only half the size. The computational overhead is only two multiplications and one addition and thus can be ignored. Note that the values u and v can be computed once and for all and then used for inverting different values c.

10.1 Wiener's Attack Against Small Exponents d

In this section we describe Wiener's attack against RSA if the exponent d is small, more precisely, if we have $d < \frac{1}{3}N^{0.25}$. Let $N = p \cdot q$ as usual and assume that $q < p < 2q$. As we have argued before, it is tempting for Bob to choose a small d, as this speeds up the inversion of RSA.

Since $ed \equiv 1 \bmod \varphi(N)$, there is an integer k such that $ed - k\varphi(N) = 1$. Thus we have

$$\left| \frac{e}{\varphi(N)} - \frac{k}{d} \right| = \frac{1}{d\varphi(N)}.$$

From $N = pq > q^2$ it follows $q < \sqrt{N}$, hence

$$0 < N - \varphi(N) = p + q - 1 < 2q + q - 1 < 3q < 3\sqrt{N}.$$

Now we obtain

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

Since $k < d$ and $d < \frac{N^{0.25}}{3}$, we have that $3k < 3d < N^{0.25}$, and hence

$$\left|\frac{e}{N}-\frac{k}{d}\right| < \frac{1}{dN^{0.25}} < \frac{1}{3d^2}. \tag{10.1}$$

Note that the fraction $\frac{e}{N}$ can be computed based on public information. Thus, this means that the secret fraction $\frac{k}{d}$ is very close to the public fraction $\frac{e}{N}$ the adversary can compute. It turns out that in this case, the adversary can even compute $\frac{k}{d}$ efficiently using the so-called continued fraction expansion.

10.1.1 Continued Fractions

We give a brief introduction on continued fractions, only as far as we will need them to show how Wiener's attack works. A *(finite) continued fraction (expansion)* is an m-tuple of non-negative integers

$$[q_1, \ldots, q_m],$$

which is an abbreviation of the following rational number:

$$q_1 + \frac{1}{q_2 + \frac{1}{q_3 + \ldots + \frac{1}{q_m}}}.$$

For $1 \leq j \leq m$, the tuple $[q_1, \ldots, q_j]$ is said to be the j-th convergent of $[q_1, \ldots, q_m]$.

Suppose a and b are positive integers such that $\gcd(a, b) = 1$, then the continued fraction expansion $[q_1, \ldots, q_m]$ of $\frac{a}{b}$ can be computed using the Euclidean algorithm. Here the quotients arising in the algorithm when applied to a and b yield the desired m-tuple. This is best demonstrated by means of an example. Assume we would like to compute the continued fraction expansion of $\frac{30}{53}$. Then applying the Euclidean algorithm to 30 and 53 yields:

$$\begin{aligned}30 &= 0 \cdot 53 + 30 \\ 53 &= 1 \cdot 30 + 23 \\ 30 &= 1 \cdot 23 + 7 \\ 23 &= 3 \cdot 7 + 2 \\ 7 &= 3 \cdot 2 + 1 \\ 2 &= 2 \cdot 1 + 0\end{aligned}$$

Then the continued fraction expansion of $\frac{30}{53}$ is the tuple $[0,1,1,3,3,2]$. Correctness of this method of computing the tuple can be checked by a simple calculation:

$$[0,1,1,3,3,2] = 0 + \frac{1}{1+\frac{1}{1+\frac{1}{3+\frac{1}{3+\frac{1}{2}}}}} = \frac{1}{1+\frac{1}{1+\frac{1}{3+\frac{2}{7}}}} = \frac{1}{1+\frac{1}{1+\frac{7}{23}}} = \frac{1}{1+\frac{23}{30}} = \frac{30}{53}$$

The following famous theorem, which we give without proof, will later allow us to break RSA for small values of d.

Theorem 10.1 *Let $a,b,c,d \in \mathbb{N}$. Assume that $\gcd(a,b) = \gcd(c,d) = 1$ and*

$$\left|\frac{a}{b} - \frac{c}{d}\right| < \frac{1}{2d^2}.$$

Then $\frac{c}{d}$ equals one of the convergents of the continued fraction expansion of $\frac{a}{b}$.

10.1.2 Wiener's Attack

Now we exploit Theorem 10.1 to mount a successful attack against RSA if d is sufficiently small. Equation 10.1 immediately allows us to apply Theorem 10.1 since $\gcd(k,d) = 1$ by assumption and since $\gcd(e,N) \neq 1$ would immediately allow us to factor N. Thus, one of the convergents of the continued fraction expansion of the known fraction $\frac{e}{N}$ equals the unknown and searched fraction $\frac{k}{d}$. These prefixes can easily be computed, but it remains to define a suitable check to identify which convergent is the desired one.

For every convergent $\frac{x}{y}$, we compute $M := (ey-1)/x$. Note that if $\frac{x}{y} = \frac{k}{d}$, then $M = \varphi(N)$. We then try to factor N given N and M using the method for factoring N and $\varphi(N)$. If we picked the correct convergent, i.e., if we indeed have $M = \varphi(N)$, this procedure will give us the correct prime factors p and q of N; for wrong convergents, no solution will exist.

Let us conclude with an illustration of this attack for artificially small parameters: Suppose $N = 160523347$ and $e = 60728973$. The continued fraction expansion of $\frac{e}{N}$ is given by $[0,2,1,1,1,4,12,102,1,1,2,3,2,2,36]$; it's first convergents are 0, $\frac{1}{2}$, $\frac{1}{3}$, $\frac{2}{5}$, $\frac{3}{8}$, $\frac{14}{37}$, and so on. For $\frac{14}{37}$, we obtain $M = \frac{60728973 \cdot 37 - 1}{14}$. This yields $p = 12347$, $q = 13001$, and indeed $p \cdot q = N$. Thus $d = 37$.

Chapter 11

Digital Signature Schemes

Digital signature schemes are the electronic equivalent of handwritten signatures, i.e., they allow distinguished principals to sign messages in such a way that the validity of the signatures can be publicly checked while additionally ensuring that nobody else is able to produce valid signatures. Technically, digital signatures can thus be seen as an asymmetric variant of MACs: While knowing the secret key is a necessary prerequisite to verify the validity of a MAC, signature schemes enable signature verification based on publicly available keys.

Digital signatures are used in various contexts and applications in practice, e.g., in electronic payment systems and for issuing certificates. In fact, digital signatures might even be of higher importance than encryption schemes in practice, as violations of integrity often turn out to be the most important concern.

11.1 Definition of Digital Signature Schemes

We now formalize the notion of digital signature schemes, or short *signature schemes*. Note that the definition closely resembles the notion of MACs, expressed in the asymmetric scenario.

Definition 11.1 (Signature Schemes) *A* signature scheme $\mathcal{I}$ *consists of three algorithms* $(\mathsf{K}, \mathsf{S}, \mathsf{V})$ *that are defined as follows:*

- *The* key generation algorithm K *is a randomized algorithm that takes no input and returns a pair* (pk, sk) *of keys, called* public key *and* private key*, respectively.*
- *The* signing algorithm S *takes a secret key* sk *and a message* m *and returns a tag* t*. This algorithm may be randomized or stateful.*
- *The* verification algorithm V *takes a public key* pk*, a message* m*, and a tag* t*, and returns* $b \in \{0,1\} \cup \{\perp\}$*. We say* V *accepts if it outputs* 1*, and it rejects otherwise. This algorithm is necessarily deterministic.*
- *The* message space $\mathcal{M}_{pk}$ *for a public key* pk *is the set of all* m *such that* $\mathsf{S}(sk, m)$ *does not output a distinguished error symbol* $\perp$ *for all* sk *with* $(pk, sk) \in [\mathsf{K}]$*.*

The above algorithms have to fulfill the following property, also referred to as the correctness *of the signature scheme: For any key-pair* $(pk, sk) \in [\mathsf{K}]$*, any message* $m \in \mathcal{M}_{pk}$*, and any* $t \in [\mathsf{S}(sk, m)]$*, we have* $\mathsf{V}(pk, m, t) = 1$*.*

Most practical signature schemes rely on the message space $\mathcal{M}_{pk} = \{0,1\}^{n_0}$ for some value n_0, or they allow the signing of arbitrary strings, i.e., $\mathcal{M}_{pk} = \{0,1\}^*$.

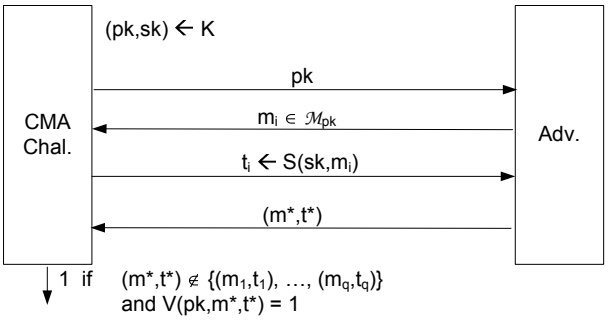

Figure 11.1: CMA-Security of Signature Schemes

11.2 CMA-Security of Signature Schemes

Security of signature schemes against existential forgery under chosen-message attack (CMA) is then defined similarly to the corresponding notion for MACs. The corresponding challenger is called a CMA challenger for signatures, but we simply speak of a CMA challenger if confusion with a CMA challenger for MACs can be excluded from the context.

Definition 11.2 (CMA Challenger for Signatures) *Let $\mathcal{I} = (\mathsf{K}, \mathsf{S}, \mathsf{V})$ be a signature scheme. The* CMA challenger *for $\mathcal{I}$ is defined as follows:*

- *First, it creates keys $(pk, sk) \leftarrow \mathsf{K}$ and outputs pk.*
- *Second, it receives messages $m_i \in \mathcal{M}_{pk}$ and outputs $t_i \leftarrow \mathsf{S}(sk, m_i)$. It repeats this stage until the input satisfies the condition of the next stage, i.e., a message-tag pair instead of a single message is input. This stage yields a sequence of pairs $(m_1, t_1), \ldots, (m_q, t_q)$.*
- *Finally, it receives a pair (m^*, t^*). If t^* is a valid tag for m^* and this pair is not contained in the above mentioned sequence, i.e., if $\mathsf{V}(pk, m^*, t^*) = 1$ and $\forall i \in \{1, \ldots, q\}\colon (m_i, t_i) \neq (m^*, t^*)$, then the CMA challenger outputs 1, and 0 otherwise.*

In the following, $Exp^{\mathsf{CMA}}_{\mathcal{I},\mathsf{A}}$ denotes the experiment where the adversary A interacts with the CMA challenger, and we let $Exp^{\mathsf{CMA}}_{\mathcal{I},\mathsf{A}} = b$ denote the event that the CMA challenger finally outputs the bit b. The advantage can now be defined as usual.

The game between the CMA challenger and an adversary is depicted in Figure 11.1. The adversary wins the game if it is able to compute a tuple (m^*, t^*) such that the challenger outputs 1. This tuple is called an *existential forgery*. Another type of forgery is a *selective forgery*, where an adversary has to be able to find a suitable tag t' *for every* (given) message m'.

Definition 11.3 (CMA Advantage) *Let $\mathcal{I} = (\mathsf{K}, \mathsf{S}, \mathsf{V})$ be a signature scheme. The* advantage *of an adversary A against the CMA challenger for $\mathcal{I}$ is defined as follows:*

$$Adv^{\mathsf{CMA}}_{\mathcal{I},\mathsf{A}} := \mathsf{Pr}\left[Exp^{\mathsf{CMA}}_{\mathcal{I},\mathsf{A}} = 1\right].$$

As usual, security is defined for a sequence of signature schemes $\mathcal{I} = (\mathcal{I}_n)_{n\in\mathbb{N}} = (\mathsf{K}_n, \mathsf{S}_n, \mathsf{V}_n)_{n\in\mathbb{N}}$ and a sequence of adversaries $\mathsf{A} = (\mathsf{A}_n)_{n\in\mathbb{N}}$. We define

$$Adv^{\mathsf{CMA}}_{\mathcal{I},\mathsf{A}}(n) := Adv^{\mathsf{CMA}}_{\mathcal{I}_n,\mathsf{A}_n}.$$

As for public-key encryption schemes, key generation as well as the signing and verification algorithms are typically defined for arbitrary values of n. Thus a (uniform) sequence of signature

schemes is typically given by single algorithms K, S, and V that take a security parameter n as input, i.e., $(\mathcal{I}_n)_{n\in\mathbb{N}} = (\mathsf{K}_n, \mathsf{S}_n, \mathsf{V}_n)_{n\in\mathbb{N}} = (\mathsf{K}(n), \mathsf{S}(n), \mathsf{V}(n))_{n\in\mathbb{N}}$.

Definition 11.4 (CMA-Secure Signature Schemes) *A sequence of signature schemes* $\mathcal{I} = (\mathcal{I}_n)_{n\in\mathbb{N}} = (\mathsf{K}_n, \mathsf{S}_n, \mathsf{V}_n)_{n\in\mathbb{N}}$ *is* secure against existential forgery under chosen-message attack (CMA)*, or simply* strongly unforgeable*, if* $Adv^{\mathsf{CMA}}_{\mathcal{I},\mathsf{A}}(n)$ *is negligible for all efficient adversaries* $\mathsf{A} = (\mathsf{A}_n)_{n\in\mathbb{N}}$.

11.3 On Signing Message Digests

Signature schemes are almost always used in conjunction with a hash function $\mathsf{H}\colon \{0,1\}^* \to \mathcal{D}$, where $\mathcal{D}$ is a subset of the set of messages that can be signed. Given a message m, the signature algorithm first applies the hash function yielding $d := \mathsf{H}(m) \in \mathcal{D}$, and then signs this message digest yielding the signature. To verify a signature one computes $d := \mathsf{H}(m) \in \mathcal{D}$ and uses the verification algorithm for this digest.

The hash function used in this design necessarily has to be collision-resistant, as otherwise the following attack can be carried out. First, find a collision for the hash function, i.e., $m \neq m'$ with $\mathsf{H}(m) = \mathsf{H}(m')$. Then let the challenger sign m. This signature is a valid signature both for m and m', as their hash values are the same. Thus both values pass verification regardless of the concrete verification algorithm. Note that this attack in practice would typically require that an attacker can come up with two *meaningful* messages that hash to the same value. On the other hand, for MD5 it took roughly one year from finding the first collision to efficiently finding two postscript files that hash to the same value. Thus it is risky to hope that finding a collision of two meaningful messages is much harder than finding any collision.

Depending on the structure of the verification algorithm the hash function may also need to be one-way. This holds, e.g., for the PKCS#1 signature scheme that we will present below.

11.4 Well-known Signature Schemes

This section is devoted to several well-known signature schemes. We start with so-called one-time signatures which can only be used to sign a single message. While this clearly constitutes a severe constraint in practice, the corresponding schemes are interesting from a conceptual point of view, and they sometimes are even sufficient for practical applications. After that, we move on to the ElGamal signature scheme and variations thereof, whose security are based on the discrete logarithm problem. Finally, we consider signature schemes based on trapdoor permutations, e.g., based on RSA. This class of signature schemes in particular contains the full-domain hash, which constitutes an efficient scheme that can be proven CMA-secure provided that trapdoor permutations exist; however the proof is in the so-called random oracle model and thus should only be considered a heuristic.

11.4.1 One-time Signatures

We review one of the first one-time signature schemes which is due to Lamport. It is interesting from a theoretical point of view since it shows that secure one-time signature schemes can be achieved only assuming the existence of one-way functions. We further sketch below how this scheme can be extended to allow signing of multiple messages.

Let us define Lamport's one-time signature scheme:

Definition 11.5 (Lamport's one-time signature) *Let $F = (\mathsf{F}(pk,\cdot))_{pk\in[\mathsf{K}(n)]}$ be a family of keyed one-way functions with key generation algorithm K and domains $\mathcal{M}_{pk}$, where $n \in \mathbb{N}$ denotes the security parameter as usual. Then Lamport's signature scheme $\mathcal{I}^{\mathsf{LS}} = (\mathsf{K},\mathsf{S},\mathsf{V})$ is defined as*

- *The* key generation algorithm K *creates a public key $pk' \leftarrow \mathsf{K}(n)$, randomly chooses $y_{i,j} \leftarrow_{\mathcal{R}} \mathcal{M}_{pk}$, and sets $z_{i,j} := \mathsf{F}(pk', y_{i,j})$ for $1 \le i \le n$ and $0 \le j \le 1$. The public key pk is then defined as $pk := (pk', (z_{i,j})_{1\le i\le n, 0\le j\le 1})$; the private key sk is defined as $sk := ((y_{i,j})_{1\le i\le n,0\le j\le 1})$.*
- *The* signature algorithm S *creates the signature sig of a message $m = m_1\cdots m_n \in \{0,1\}^n$ with respect to a secret key $sk := ((y_{i,j})_{1\le i\le n,0\le j\le 1})$ by computing*

$$sig := (y_{1,m_1}, \ldots, y_{n,m_n}).$$

- *The* verification algorithm V *verifies a signature $sig = (s_{1,m_1}, \ldots, s_{n,m_n})$ on a message $m = m_1\cdots m_n \in \{0,1\}^n$ with respect to a public key $pk := (pk', (z_{i,j})_{1\le i\le n,0\le j\le 1})$ by computing*

$$\mathsf{V}(pk, m, sig) := \begin{cases} 1 & \text{if } \mathsf{F}(pk', s_{i,m_i}) = z_{i,m_i} \text{ for all } 1 \le i \le n, \\ 0 & \text{otherwise.} \end{cases}$$

It can easily be seen that a correctly generated signature passes verification. Indeed, let $y_{i,j}$, $z_{i,j}$, and pk' be as constructed by the key generation algorithm, and let $(s_1\cdots s_n)$ be a signature for the message $(m_1\cdots m_n)$. Then $s_i = y_{i,m_i}$ holds by construction, and thus by construction of the $z_{i,j}$ we have $z_{i,m_i} = \mathsf{F}(pk', s_{i,m_i})$, thus passing the verification algorithm.

11.4.2 ElGamal Signatures

The ElGamal signature scheme was proposed in 1985, i.e., roughly at the same time the ElGamal encryption scheme was invented. A modified version of the ElGamal signature scheme was later adopted as the Digital Signature Algorithm (DSA) which we will describe in the next section. Similar to the ElGamal encryption scheme, the ElGamal signature scheme is probabilistic, i.e., signing a message multiple times will yield different signatures.

Definition 11.6 (ElGamal Signatures) *The ElGamal signature $\mathcal{I}^{\mathsf{ElGamal}} = (\mathsf{K},\mathsf{S},\mathsf{V})$ scheme is defined as follows:*

- *The* key generation algorithm K *randomly chooses an n-bit prime p, a generator g of $\mathbb{Z}_p^*$, and an integer $x \in \{0,\ldots,p-2\}$. It then sets $h := g^x$. The public key and the secret key are $pk := (p,g,h)$ and $sk := (p,g,x)$, respectively. The message space is $\mathcal{M}_{pk} := \{0,\ldots,p-2\}$.*
- *The* signature algorithm S *creates the signature sig of a message $m \in \mathcal{M}_{pk}$ with respect to a secret key $sk = (p,g,x)$ by choosing a random number $r \leftarrow_{\mathcal{R}} \{1,\ldots,p-1\}$ with $\gcd(r, p-1) = 1$ and computing*

$$\begin{aligned} s &:= g^r \bmod p, \\ t &:= (m - xs)r^{-1} \bmod (p-1), \\ sig &:= (s,t). \end{aligned}$$

- *The* verification algorithm V *verifies a signature $sig = (s,t)$ on a message m with respect to a public key $pk = (p,g,h)$ by computing*

$$\mathsf{V}(pk, m, (s,t)) := \begin{cases} 1 & \text{if } h^s s^t \equiv g^m \bmod p, \\ 0 & \text{otherwise.} \end{cases}$$

The correctness of ElGamal signatures can be seen as follows: Let $(s,t) \in [\mathsf{S}(sk,m)]$ be a signature for pk and sk as defined above. Then we have

$$\begin{aligned}
\mathsf{V}(pk,m,(s,t)) = 1 &\Leftrightarrow h^s \cdot s^t \equiv g^m \pmod p \\
&\Leftrightarrow (g^x)^s \cdot s^{(m-xs)r^{-1}} \equiv g^m \pmod p \\
&\Leftrightarrow (g^x)^s \cdot (g^r)^{(m-xs)r^{-1}} \equiv g^m \pmod p \\
&\Leftrightarrow g^{xs+r(m-xs)r^{-1}} \equiv g^m \pmod p \\
&\Leftrightarrow xs + (m-xs) \equiv m \pmod{p-1} \\
&\Leftrightarrow m = m \in \mathcal{M}_{pk}.
\end{aligned}$$

Now we take a look at the security of the ElGamal signature scheme. Let us first note that the security of the ElGamal signature scheme has not been proven; in fact, the ElGamal signature scheme is not CMA-secure in the form it is stated above. We will first give some indication why selectively forging signatures could be as hard as computing discrete logarithms. Suppose Eve tries to forge a message m without knowing the secret value x. If she chooses s first and then tries to find t such that (s,t) is a signature for m, then she has to find $t = \mathsf{DLog}_s(g^m h^{-s})$, i.e., to compute a discrete logarithm. If she chooses t first, then she has to solve the equation $h^s s^t \equiv g^m \pmod p$ for s. For this, no feasible solution is known either; however, it cannot be reduced to the discrete logarithm problem. There might even be the possibility that one can compute s and t simultaneously, but no-one has discovered a way to do this so far.

Things change considerably if an attacker is satisfied with creating an existential forgery, which we consider as the standard threat for signature schemes. We will see in the next paragraph that existential forgeries can be computed efficiently for the ElGamal signature scheme.

Existential Forgeries for the ElGamal Signature Scheme. The ElGamal signature scheme as described above is existentially forgeable as follows. Suppose we know the public key, i.e., we know g and h, and we want to find a signature $sig = (s,t)$ which passes the verification algorithm. Fix integers i,j such that $1 \le i,j \le p-2$, and let $s = g^i h^j$. Then the verification condition is $g^m \equiv h^s (g^i h^j)^t \pmod p$, or, equivalently, $g^{m-it} \equiv h^{s+jt} \pmod p$. The later congruence is satisfied if $m - it \equiv 0 \pmod{p-1}$ and $s + jt \equiv 0 \pmod{p-1}$. We are given i and j, thus, provided that $\gcd(j, p-1) = 1$, we can easily solve these congruences for t and m, obtaining $t = -sj^{-1}$ and $m = it = -sij^{-1}$. The construction immediately implies that (s,t) is a valid signature for m. This attack can be countered by applying a hash function to m before it is input to the ElGamal signature scheme.

11.4.3 Digital Signatures Algorithm (DSA)

The Digital Signature Algorithm (DSA) was proposed by the NIST in 1991 for the Digital Signature Standard (DSS). Some criticisms however arose when the standard was proposed: One (yet minor) issue was the "closed-door" approach that was employed in the process which did not involve the U.S. industry. The more severe point of criticism was that the size of the modulus p was initially fixed to 512 bits. The standard was later updated to allow moduli p of larger size, and in 2001 the NIST itself recommended the use of 1024-bit primes p.

The Digital Signature Algorithm is defined as follows:

Definition 11.7 (Digital Signatures Algorithm (DSA)) *Let* $\mathsf{H}\colon \{0,1\}^{2^{64}-1} \to \{0,1\}^{160}$ *be the SHA-1 hash function, and let* L *such that* $L \equiv 0 \pmod{64}$ *and* $512 \leq L \leq 1024$. *The DSA signature scheme* $\mathcal{I}^{\mathsf{DSA}} = (\mathsf{K}, \mathsf{S}, \mathsf{V})$ *is defined as follows:*

- *The* key generation algorithm K *randomly chooses an* L*-bit prime* p *and a* 160*-bit prime* q *such that* $q \mid p-1$. *It then randomly chooses* $g \in \mathbb{Z}_p^*$ *of order* q *and* $x \leftarrow_{\mathcal{R}} \{0, \ldots, q-1\}$. *Finally, it sets* $h := g^x$. *The public key and the secret key are then defined as* $pk = (q, p, g, h)$ *and* $sk = (q, p, g, x)$, *respectively.*
- *The* signature algorithm S *creates a signature* sig *on a message* $m \in \{0,1\}^{2^{64}-1}$ *with respect to a secret key* $sk = (q, p, g, x)$ *by choosing a random* $r \leftarrow_{\mathcal{R}} \{1, \ldots, q-1\}$ *and by computing*

$$\begin{aligned} s &:= (g^r \bmod p) \bmod q, \\ t &:= (\mathsf{H}(m) + xs) r^{-1} \bmod q, \\ sig &:= (s, t). \end{aligned}$$

  If either s *or* t *equal* 0, *a new* r *is chosen and the signature is recomputed. (This happens only with very small probability and thus does not raise problems in practice).*
- *The* verification algorithm V *verifies a signature* $sig = (s, t)$ *on a message* m *with respect to a public key* $pk = (q, p, g, h)$ *by computing*

$$\mathsf{V}(pk, m, (s,t)) := \begin{cases} 1 & \text{if } (g^{\mathsf{H}(m)t^{-1} \bmod q} h^{st^{-1} \bmod q} \bmod p) \bmod q = s, \\ 0 & \text{otherwise.} \end{cases}$$

Correctness of the scheme can be shown as usual.

11.4.4 RSA-based Signature Schemes

In this section we describe a simple construction of a signature scheme based on the RSA trapdoor permutation. This construction can be found in various books; however, it can easily be broken by various attacks. Later we will see how these attacks can be countered by using hash functions and by including redundancy inside the signature, as it is done for example in the PKCS#1 standard.

Definition 11.8 (Naive RSA) *The naive RSA signature scheme works as follows:*

- *Keys are generated exactly as for (naive) RSA encryption: Randomly choose two* n*-bit primes* p *and* q, *let* $N := pq$, *and pick* e, d *such that* $ed = 1 \bmod \varphi(N)$. *Let* $pk = (N, e)$ *be the public key, and let* $sk = d$ *be the secret key.*
- *To sign a message* $m \in \{1, \ldots, N-1\}$ *with respect to the secret key* $sk = d$ *one computes*

$$\mathsf{S}(sk, m) := m^d \bmod N.$$

- *A signature* sig *of a message* m *with respect to the public key* pk *is verified by computing*

$$\mathsf{V}(pk, m, s) := \begin{cases} 1 & \text{if } sig^e \equiv m \pmod{N}, \\ 0 & \text{otherwise.} \end{cases}$$

Correctness of the scheme is a direct consequence of Lemma 9.1.

Attacks against Naive RSA Signatures. There are several attacks against the above signature scheme. The first attack is an existential forgery under a passive attack. One picks an arbitrary $sig \in \mathbb{Z}_N$ and computes $m := sig^e \bmod N$. Then sig is a signature for the message m, as one can easily verify.

A more sophisticated, active attack is to create selective forgeries using blinding techniques similar to those we have seen earlier. Suppose the adversary wants to get a signature for a message m. He chooses a random $r \in \mathbb{Z}_N^*$ and computes $m' := m \cdot r^e \bmod N$. He asks the signer to sign m', yielding a signature $s' = (m')^d \bmod N$. Finally, he computes $s := s'/r$. Then s is a valid signature for m, as the following computation shows:

$$s^e = \left(\frac{s'}{r}\right)^e = \left(\frac{(m')^d}{r}\right)^e = \left(\frac{(mr^e)^d}{r}\right)^e \equiv \left(\frac{m^d \cdot r}{r}\right)^e \equiv \left(m^d\right)^e \equiv m \pmod N.$$

Note that the ability to let a message be signed blindly can even be a desirable feature which is used in a cryptographic primitive called *blind signature*. This is used whenever it is desirable that the signer should not learn what he is actually signing, e.g., this preserves anonymity in electronic payment systems. We will see this in more detail later in the course.

Adding Redundancy. One possibility to counter these blinding attacks is to add redundancy to the message before applying the RSA signature operation, i.e., one computes $sig := (\mathsf{red}(m), m)^d \bmod N$, and verifies the validity of the signature by testing $sig^e = (\mathsf{red}(m), m) \bmod N$. Now what should this padding look like? A bad way to do it is to prepend zeroes, as this enables mainly the same attacks as before. A good solution is to prepend something "chaotic", e.g., the encryption of the message m under DES with a fixed (public) key. Then a signature needs to be rejected if the padding is not of the correct form. This padding is used in an ISO standard. The padding in the PKCS#1 standard is similar: There, a fixed prefix of the following form is prepended to the message (in hexadecimal notation)

$$0001\ 0101\ 0101\ \ldots\ 0101\ 0000 \mid\mid \mathsf{H}(m).$$