Attacks on RSA cryptosystem

1 The attacker knows the modulus n and its totient value $\phi(n)$

There are several ways in which the value of $\phi(n)$ might be guessed by a clever attacker if one is not careful with implementing the RSA system – as will be seen below. The attacker knows that n is a product od two primes, say p and q, and that its totient value is $\phi(n) = (p-1)*(q-1)$, but does not a priori know the values of the prime numbers p and q.

Now $\phi(n) = p*q-(p+q)+1 = n-(p+q)+1$, and so $q = (n-\phi(n)+1)-p$.

Substituting this into the equation of n we find $n = p*((n-\phi(n)+1)-p)$, and thus

$$p^2 - p*(n-\phi(n)+1)+n = 0$$

The only unknown in this quadratic equation is p. Now using the quadratic formula, p's value can befound. Then from the value of p and n we can find the value of q.

Defense: Guard $\phi(n)$ as carefully as p, q or the private key t.

2 Common modulus attack

Consider the following scenario:

The Information Security Department (ISD) of an organization issues each of the organization's employees, domestic and abroad, with an individual RSA key. To simplify the PKI (public key infrastructure) maintenance for the organization, it was decided that a single modulus n will be generated by the ISD and each employee will be issued a personalized public encryption exponent e, and corresponding private decryption exponent d. All key-generating computations will be done by the ISD. This way the organization will also have a copy of each employee's RSA key-set. The reasons for this were given as:

- When an employee loses a key, the ISD can always reissue a key;
- When an internal security leak is suspected, the ISD will have all keys available to investigate the incident;
- When law enforcement obtains a "supina" for a particular employee's RSA keys, a central office - the ISD - can see to it that the "supina" is honored.

Suppose further that the ISD decides that the encryption exponents will be chosen so that the *gcd* of any pair is 1; for example, to speed up selecting values of encryption exponent e, the ISD selects from a database of prime numbers.

An eavesdropper can under certain circumstances decrypt messages sent in this system without needing access to any private keys:

Suppose that the CEO dispatches the same message M to two different office managers whose encryption exponents respectively are e_1 and e_2. The encrypted messages are:

$$E = M^{e_1} \quad mod \quad n \text{ and } F = M^{e_2} \quad mod \quad n$$

An eavesdropper has access to the public keys n, e_1 and e_2, and the encrypted messages E and F. Since $gcd(e_1, e_2) = 1$, the eavesdropper applies the extended Euclidean algorithm to compute integers x and y such that $x * e_1 + y * e_2 = 1$. Then with x and y in hand, the eavesdropper computes

$$E^x * F^y \quad mod \quad n$$

Observe that this works out to be M, and the eavesdropper decrypted the message!

One defense against this attack would be to ensure that the set of all encryption exponents has a number larger than 1 as a common factor. Though this might defend sufficiently against an outside eavesdropper, we shall see next that it is no defense against an insider attempting to steal proprietary information from the organization.

3 The public encryption exponent is “small”

Consider the previous scenario with the single modification that the ISD does not impose additional requirements on the encryption exponents, except that no two employees be issued the same one (an obvious requirement). Thus, each employee has the following information: The value of n, the employee’s personal public encryption exponent e, and the employee’s personal private decryption exponent d. It often happens in practice that at least the encryption exponent e generated by implementations of RSA are ”small”. For example: For PGP (Pretty Good Privacy) the great majority of e’s generated by the software is 17, for Secure Shell (SSH) the great majority of e’s generated is 37, and the US Postal Services’ prescription for RSA implementations to handle e-stamps requires the encryption exponent to be 65537. Suppose that this is also the case in this scenario - e is ”small”. Now the malicious insider has the following method for factoring n using the issued public value of e and private value of d. Notice that $e * d = 1 \ \ mod \ \ \phi(n)$.

Now $\phi(n)$ is not a priori known to the employees, but it is known that for some natural number $k < \min\{e, d\}$ such that $e * d = 1 + k * \phi(n)$.

Thus, there is an integer $k < min\{e, d\}$ such that $\phi(n) = (e * d - 1)/k$.

This suggests a search procedure for discovering $\phi(n)$, with the length of the list of possible values shorter than $min\{e, d\}$. Once the correct value of $\phi(n)$ is discovered, n can be factored using techniques discussed in the Section 1. Indeed, once $\phi(n)$ is discovered, all the private keys of all the employees can be computed from their known public key, and then all correspondence to them can be decrypted.

4 Common encryption exponent

Some popular RSA encryption implementations (for example PGP and SSH) have the attribute that a great majority of the generated encryption exponents have the same value (for PGP it is 11, while for SSH it is 37). When such a system is used to send a common message to a large number of key owners, then there is a method of decrypting the message without discovering the private key. This attack is based on the Chinese Remainder Theorem.

To be precise, assume that a common value for the encryption exponent is shared by k different users, and let $R_1, R_2, ..., R_k$ denote their RSA encryption moduli.

We may assume that for $i < j$ we have$gcd(R_i, R_j) = 1$; otherwise we either discover a nontrivial factor of these moduli and thus could compute their private keys, or else we would have two owning exactly the same key.

Now consider a message, denoted M but not a priori known to us.

Suppose that this message is encrypted for all these key owners. Also assume that $M < R_i$ for each i. Now the k encrypted versions are:

$$E_i = M_k \quad mod \quad R_i,\ i = 1, ..., k.$$

According to the Chinese Remainder Theorem there is a unique $x < R_1 * R_2 * ... * R_k$ such that for each i we have $x = E_i \quad mod \quad R_i$.

Also, the theorem describes exactly how to compute x from the data. But $M_k < R_1 * R_2 * ... * R_k$, and also satisfies these equations. Thus, by its uniqueness, $x = M_k$.

Now we compute M by taking the ordinary k-th root of x.

5 Wiener's attack

It is sometimes tempting to choose the private decryption exponent d small to speed up the decryption process. A small value of d always leads to a large value of e, the public encryption exponent, and thus to slower encryption. There are two reasons to avoid choosing a small private decryption exponent. One of them is that correspondents will be discouraged by the "slow" encryption process and will most likely not communicate. The other is more ominous – it opens up the system to a fairly efficient attack. The second is the subject of this section. Recall how the Euclidean Algorithm computes the greatest common divisor, g, of two numbers a and b with $a < b$. One obtains a list of equations using long division:

$$b = q_1 * a + r_1, \qquad 0 \leq r_1 < b$$

$$a = q_2 * r_1 + r_2, \qquad 0 \leq r_2 < r_1$$

$$r_1 = q_3 * r_2 + r_3, \qquad 0 \leq r_3 < r_2$$

$$r_2 = q_4 * r_3 + r_4, \qquad 0 \leq r_4 < r_3$$

$$\vdots$$

$r_{n-1} = q_{n+2} * r_{n+1} + r_{n+2}$ and $r_{n+2} = 0$ while $r_{n+1} > 0$.

The last non-zero remainder, r_{n+1}, is $gcd(a, b)$.

Consider a rational number b/a with $gcd(a, b) = 1$. Then $r_{n+1} = 1$. From the same set of equations we obtain:

$$
\begin{aligned}
b/a &= q_1 + (r_1/a) \\
&= q_1 + 1/(a/r_1) \\
&= q_1 + 1/(q_2 + (r_2/r_1)) \\
&= q_1 + 1/(q_2 + 1/(r_1/r_2)) \\
&= q_1 + 1/(q_2 + 1/(q_3 + r_3/r_2)) \\
&. \\
&. \\
&. \\
&= q_1 + 1/(q_2 + 1/(q_3 + 1/(q_4 + 1/(q_5 + (... + 1/q_{n+2})...)))))
\end{aligned}
$$

This last expression is known as the continued fraction expansion of b/a, and is for typographical convenience written as $b/a = [q_1, q_2, q_3, q_4, ..., q_{n+2}]$. The number $C_j = [q_1, q_2, ..., q_{j+1}]$ is said to be the **j-th convergent of b/a**. The convergents of a fraction can be computed efficiently by using the Euclidean Algorithm, and they satisfy very important relations. The one relevant for us is the following

Theorem 1 *Assume that $gcd(a, b) = 1$. If r, s are any natural numbers such that $gcd(r, s) = 1$, and $|a/b - r/s| < 1/(2s^2)$ then r/s is one of the convergents of a/b.*

Theorem 2 *Let n be an RSA modulus, say $n = p * q$ where p and q are primes, and let e be the public encryption exponent and $d$the private decryption exponent. If $d < \sqrt[4]{n}/3$ (i.e., d is "small"), $q < p < 2q$ and $ed = 1 + k\phi(n)$ (with $k < mine, d$) then $|e/n - k/d| < 1/(2d^2)$.*

Thus, by the previous theorem, k/d is a unique convergent of e/n. Note also that $gcd(k, d) = 1$. Since e/n is a publicly known quantity, the Euclidean algorithm can be used to compile its set of convergents. The set of convergents can then be examined to determine which one's denominator is d. This can be done by encrypting a phrase using n and e, and then trying to decrypt it using n and the denominator of a convergent. Once the correct denominator is found, one also knows k, and so one can use e, d and k to solve for $\phi(n)$. Subsequently n can be factored. Wiener's bound is not tight.

This result shows that if one is not careful with the selection of p, q and d, then the Euclidean Algorithm can be used to dramatically reduce the amount of work involved in a search for the private key d. Since typically n is 1024 bits, it follows that d must be at least 256 bits long in order to avoid this attack. This is unfortunate for low-power devices such as smartcards, where a small d would result in big savings. All is not lost however. Wiener describes a number of techniques that enable fast decryption and are not susceptible to his attack. One of the techiniques involves Chinese Remainder Theorem:

Suppose one chooses d such that both $d_p = d \ \ mod \ \ (p-1)$ and $d_q = d\, mod\, (q-1)$ are small, say 128 bits each. Then fast decryption of a ciphertext C can be carried out as follows: first compute $M_p = C^{d_p}\, mod \ \ p$ and $M_q = C^{d_q}\ mod\, q$. Then use the CRT to compute the unique value M satisfying $M = M^p\ mod\, p$ and $M = M^q\ mod\, q$. The resulting M satisfies $M = C^d\ mod\, n$ as required. The point is that although d_p and d_q are small, the value of $d\, mod\ \phi(n)$ can be large. As a result, the attack of Theorem 2 does not apply.

6 Iterated Encryption

The Iterated Encryption Attack is based on the Euler - Fermat theorem.

Theorem 3 *If n is a positive integer and m is a positive integer coprime to n, then $m^{\phi(n)}\ mod\, n = 1$.*

Consider a public RSA key with encryption modulus n and encryption exponent e. Also, consider a message $M < n$ in padded ASCII form. Consider what might happen when M is encrypted iteratively using this key:

$$\begin{aligned} E_1 &:= M^e \quad mod \quad n \\ E_2 &:= E_1^e \quad mod \quad n \\ E_3 &:= E_2^e \quad mod \quad n \\ &\vdots \\ E_{k+1} &:= E_k^e \quad mod \quad n \end{aligned}$$

Using the laws of exponents we have that for each n, $E_k = M^{(e^k)}\ \ mod\ n$.

Also, by the Euler - Fermat theorem, $M^{\phi(n)} = 1 \quad mod \quad n$, unless $gcd(M, n) \neq 1$ in which case we can factor n, since $M < n$. Thus, we need to be concerned only with $e^k \quad mod \quad \phi(n)$ when considering these repeated encryptions of M. Since there is no a priori bound on the number of times we might encrypt using e and n, but there is a bound on the number of possible values $mod \quad \phi(n)$ of e^k, we see that at some point these values will start repeating. Since $gcd(e, phi(n)) = 1$, we can then conclude that there is an k such that

$$e^k \ mod\ \phi(n) = 1 \text{ and thus, } E_k = M.$$

This means that merely through repeating the encryption process, we eventually decrypt the originally encrypted message. Of course, if the smallest k

at which this occur is an enormous number, this is not an efficient method for decrypting the message. But, if the designer of the key did not choose e carefully, this value of k may be relatively small, even just a few hundred or a few thousand. Small values of s are typically safe against this sort of attack, but as we have seen, there are circumstances under which one might want to have large values of e(for example when users of a system must share a common n, we want to ensure that neither e, nor d (the private key) is small enough).

When large values of e are considered, this particular attack must be considered during the design stage.Observe also that if we find the first value of k at which $E_k = M$ then we have found the first k at which $e^k \, mod \, \phi(n) = 1$ and so we could use $e^{(k-1)} \, mod \, \phi(n)$ as stand in for d.

7 Partial private key

We have seen earlier that if a number of people share a common RSA encryption modulus n, but each has their own public encryption exponent e and private decryption exponent d, then there is a method to factor n; its efficiency depends on the size of e and of d. One might think that this scenario is only an academic matter, and that one party never really has access to another's private key by mere computation, since everybody will ensure they don't share an n with anyone else. As we shall see now, depending on the choice of the prime factors p and q of n, a large portion of the private key d may anyway be leaked by the system. This emphasizes once more that the choice of the prime numbers p and q must be made very carefully, and it also emphasizes that certain secret sharing (trustee) schemas must be avoided.
A key inequality which underlies the analysis here is: $2 \cdot \sqrt{x \cdot y} \leq x + y$.
Recall that the relationship among e, d and $\phi(n)$ is:

$$e \cdot d \, mod \, \phi(n) = 1$$

which means that there is a $k < min\{e, d\}$ such that

$$e \cdot d = 1 + k \cdot \phi(n).$$

Thus, there is a $k < min\{e, d\}$ such that $d = (k \cdot \phi(n) + 1)/e$.

From the point of view of the outsider, k and $\phi(n)$ are unknown, but n and e are known. However, there is a relationship between $n = p \cdot q$ where $p > q$ are prime, and $\phi(n)$, which can be used to approximate $\phi(n)$, namely: $\phi(n) = (p-1) \cdot (q-1) = p \cdot q - (p+q) + 1 < p \cdot q - 2 \cdot \sqrt{p \cdot q} + 1 = n - 2 \cdot \sqrt{n} + 1$, by the beginning inequality. Using the quantity on the right, which is computable from n alone, as an approximation for $\phi(n)$, we define for $k < d$ as follows:

$$d_k := (k \cdot (n - 2 * \cdot \sqrt{n} + 1) + 1)/e.$$

Then d_k is the attacker's first approximation of the private d. If the attacker also has the least significant digits of d, say L, then the attacker can improve the approximation d_k by replacing the corresponding least significant digits of d_k by the digits of L. Let the resulting improvement be D_k. The attacker's guess will be this D_k. Observe that for the correct value of k, $d_k \geq d$. To see how close d_k is to the actual d, compute

$$|d_k - d| = (k/e) \cdot (\sqrt{p} - \sqrt{q})^2.$$

Since $k < e$, we find that for the correct k,

$$(1/e) \cdot (\sqrt{p} - \sqrt{q})^2 \leq |d_k - d| < (\sqrt{p} - \sqrt{q})^2.$$

The smaller the value of $(\sqrt{p} - \sqrt{q})^2$, the quicker the attacker will arrive at a correct guess for the value of d. One finds that in any case $|d_k - d| < p$, and so, if p and q have the same number of digits, then half of the most significant digits of d coincide with those of one of the d_k's. What this indicates is that for certain choices of p and q the following trustee schema is dangerous: Two partners in a business decide that neither will have access to the private key d of the business, but instead they will each have a share of the key. This is to ensure that either can read encrypted confidential information only with the collaboration of the other. If one of them gets the least significant half of d, and the other gets the most significant half of d, then the one holding the least significant half has a distinct advantage in reconstructing the whole private key - especially if the prime numbers p and q have the same number of digits.

8 Wiener's covert channel

To see how the channel works consider RSA key generation scheme. In Section 5 we discussed Wiener's attack and the conditions under which an RSA key is weak against Wiener's attack. Now we will use this knowledge to install covert channel in the key generation algorithm.
Namely the algorithm chooses primes p and q and private key d as in Theorem 2. Then chooses a fixed number P and computes $e_2 = e_1 \cdot Pmodn$ where $e_1 = 1/d \, mod \, \phi(n)$. Note that the key (n, e_1) is weak against Wiener's attack. The algorithm generates an RSA public key (n, e_2) whenever $gcd(e_2, \phi(n)) = 1$. Then it computes the RSA private key by computing $d_2 = 1/e_2 \, mod \, \phi(n)$.
The number P is called a *covert number.*

9 Blinding Attack on RSA signature

Let d be Bob's private key and (n, e) be his corresponding public key. Suppose an adversary Eve wants Bob's signature on a message M, $M < n$. Being no fool, Bob refuses to sign M. Eve can try the following: she picks a random number r and computes $M' = r^e \cdot M \, mod \, n$. She then asks Bob to sign the random message M'. Bob may be willing to provide his signature on the innocent-looking M'.

But recall that $S' = (M')^d \, mod \, n$.

Eve now computes $S = S'/r \quad mod \quad n$ and obtains Bob's signature S on the original M. Indeed, $S^e = (S')^e/r^e = (M')^{ed}/r^e = M'/r^e = M \, mod \, n$.

This technique enables Eve to obtain a valid signature on a message of her choice by asking Bob to sign a random "blinded" message. Bob has no information as to what message he is actually signing.

10 Random Faults attack on RSA

Implementations of RSA decryption and signatures frequently use the Chinese Remainder Theorem to speed up the computation of $M^d modn$. Instead of working modulo n, the signer Bob first computes the signatures modulo p and q and then combines the results using the Chinese Remainder Theorem. More precisely, Bob first computes $E_p = M^d \, mod \, p$ and $E_q = M^d \, mod \, q$ and then obtains the signature E by using the Chinese Remainder Theorem.

We described a version of the attack due to A. K. Lenstra. Suppose a single error occurs while Bob is generating a signature. As a result, exactly one of E_p or E_q will be computed incorrectly. Say E_p' is not correct, but E_q is correct. The resulting signature using the Chinese Remainder Theorem is $E' = (q \cdot x \cdot E_p + p \cdot y \cdot E_q) \, mod \, n$. Once Eve receives E', she knows it is a false signature since $V' = (E')^e \neq M \, mod \, n$. However, notice that $V' \neq M \, mod \, p$ while $V' = M \, mod \, q$. As a result, $gcd(n, (V' - M) \, mod \, n)$ exposes a nontrivial factor of n.