Public Key Cryptography

Lecture 11

Practical Aspects of RSA

Index

Security Mechanisms of Public Key Algorithms

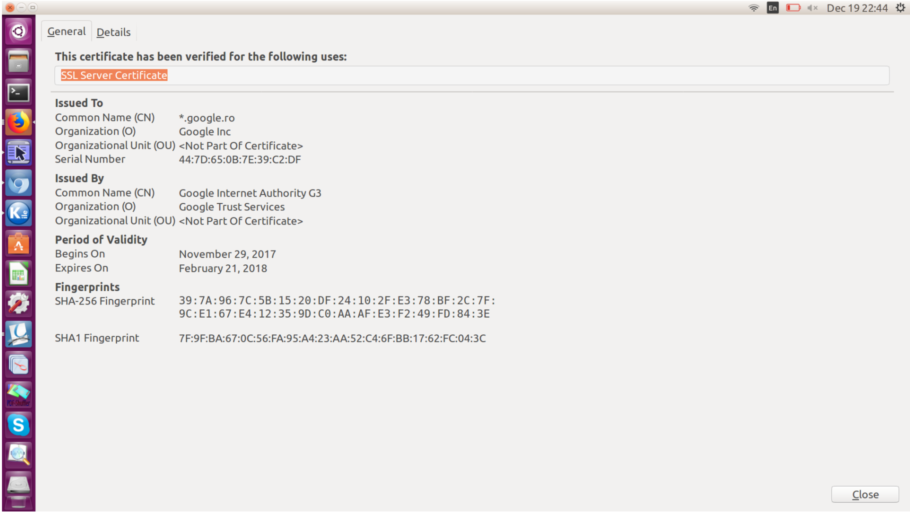
1. **Key Establishment:** protocols for establishing secret keys over an insecure channel (e.g., Diffie-Hellman key exchange).
2. **Non-repudiation and message integrity:** digital signature algorithms (e.g., RSA, DSA).
3. **Identification:** challenge-and-response protocols together with digital signatures (e.g., in applications such as smart cards for banking or for mobile phones).
4. **Encryption:** RSA, ElGamal etc.

Identification and encryption can also be done by using private key cryptography (symmetric ciphers).

Most practical protocols are hybrid (e.g. SSL/TLS), incorporating both public key and private key cryptography.

Authenticity of public keys

Certificate: bind a public key to a certain identity.



Public key cryptosystems vs. private key cryptosystems

- Advantages of public key cryptosystems
  - only the private key must be kept secret
  - the administration of keys requires the presence of only a trusted third party
  - private / public keys may remain unchanged for some time
  - digital signature schemes; the key used to describe the public verification function is quite small
  - key distribution; in a large network the number of needed keys is much smaller
- Disadvantages of public key cryptosystems
  - the speed rates for public key encryptions are several orders of magnitude slower than the best known private key encryptions
  - key sizes are much larger than those used by private key encryption, and the size of public key signatures is larger
  - no public key cryptosystem has been proven to be secure (the same can be said for block ciphers)
  - the relative recent history of public key cryptography

Main public key cryptosystems

- RSA (1977)
  - Integer Factorization Problem
- Rabin (1979)
  - Modular Square Root Problem
- ElGamal (1985)
  - Discrete Logarithm Problem
- Elliptic Curve (1985-Miller, 1987-Koblitz)
  - Elliptic Curve Discrete Logarithm Problem
- There is no known polynomial time algorithm for solving the above problems!
- Others: McEliece (linear code decoding problem), Knapsack (subset sum problem) etc.; security issues!

Key lengths and security levels

Definition

An algorithm is said to have a *security level* of n bit if the best known attack requires 2^n steps.

Algorithm Family	Cryptosystems	Security Level (bit)			
		80	128	192	256
Integer factorization	RSA	1024 bit	3072 bit	7680 bit	15360 bit
Discrete logarithm	DH, DSA, Elgamal	1024 bit	3072 bit	7680 bit	15360 bit
Elliptic curves	ECDH, ECDSA	160 bit	256 bit	384 bit	512 bit
Symmetric-key	AES, 3DES	80 bit	128 bit	192 bit	256 bit

RSA in Practice

- Choosing the modulus n
  - a 256-bit modulus can be factored in a few hours on a personal computer, using freely available software
  - the largest number factored: an 829-bit number with 250 decimal digits, announced in February 2020
  - a modulus n of at least 2048 bits is recommended
  - for long-term security, 4096-bit or larger moduli should be used
  - summary of RSA numbers factoring (part of RSA factoring challenge) records:

Decimal digits	Bit length	Year
200	663	2005
212	696	2013
220	729	2016
230	762	2018
232	768	2009
240	795	2019
250	829	2020

RSA in practice (cont.)

- Choosing the primes p and q

  The primes p and q should be selected such that factoring $n = pq$ is computationally infeasible. Main conditions:

  - p and q should be about the same bitlength, and sufficiently large. For example, if a 2048-bit modulus n is to be used, then each of p and q should be about 1024 bits in length.
  - the difference $p - q$ should not be too small. If $p \approx q$, then the (generalized) Fermat method can be successfully applied to factor n.
  - none of p and q should be small. Otherwise, the Pollard ρ method can be successfully applied to factor n.

  One randomly generates numbers of the required bit-length and test (by Miller-Rabin Algorithm) whether they are prime. E.g., for RSA with a 1024-bit modulus n, p and q each should have about 512-bit length. Then the probability that a random 512-bit odd integer is a prime is $1/177$.

RSA in practice (cont.)

- Choosing the encryption exponent e
  - If e is chosen at random, then RSA encryption using the repeated square-and-multiply algorithm takes k modular squarings and an expected $k/2$ (less with optimizations) modular multiplications, where k is the bitlength of the modulus n. It can be sped up by selecting e to be small and/or with a small number of 1's in its binary representation.
  - $e = 3$ is commonly used in practice; in this case, it is necessary that neither $p - 1$ nor $q - 1$ be divisible by 3. This results in a very fast encryption operation since encryption only requires 1 modular multiplication and 1 modular squaring.
  - $e = 2^{16} + 1 = 65537$ is also used in practice. It has only two 1's in its binary representation, and so encryption using the repeated square-and-multiply algorithm requires only 16 modular squarings and 1 modular multiplication. The encryption exponent $e = 2^{16} + 1$ has the advantage over $e = 3$ in that it resists the kind of attack on small encryption exponents, since it is unlikely the same message will be sent to $2^{16} + 1$ recipients.

RSA in practice (cont.)

- Choosing the decryption exponent d
  - One should avoid choosing a small decryption exponent d in order to avoid brute-force attacks.
  - For large d one can speed-up decryption $c^d \bmod n$ as follows. Denote

$$c_p = c \bmod p, \quad c_q = c \bmod q,$$

$$d_p = d \bmod (p-1), \quad d_q = d \bmod (q-1).$$

    Compute

$$y_p = c_p^{d_p} \bmod p, \quad y_q = c_q^{d_q} \bmod q.$$

    Now use the Chinese Remainder Theorem to get:

$$m = (qa)y_p + (pb)y_q \bmod n,$$

    where $a = q^{-1} \bmod p$ and $b = p^{-1} \bmod q$.

RSA in practice (cont.)

Padding/salting: to prevent an attacker to derive statistical properties from the ciphertext.

RSA encryption is deterministic, i.e., for a specific key, a particular plaintext is always mapped to a particular ciphertext.

An attacker can derive statistical properties of the plaintext from the ciphertext.

Also, given some pairs of plaintext-ciphertext, partial information can be derived from new ciphertexts which are encrypted with the same key.

RSA security

- Small encryption exponent e
  - to increase the encryption speed: e might be either small or with many 0's in its binary writing
  - the same small e should not be used in case the same message is sent to many users

    Indeed, if Alice sends a message m to 3 users having the same $e = 3$ and moduli n_1, n_2, n_3, then she sends $c_i = m^e \bmod n_i$, $i = 1, 2, 3$. Since the n_i's are most likely pairwise relatively prime, the intruder Eve who observes c_1, c_2, c_3 can use the Chinese Remainder Theorem to obtain a unique solution x of the system $x \equiv c_i \pmod{n_i}$, $i = 1, 2, 3$. But $m^e < n_1 n_2 n_3$, hence $x = m^e$ and $m = \sqrt[e]{x}$.
    Prevention: *salting* (attach to each message a random sequence of bits, at least 64).
  - a small e should not be used in case of short messages

    Indeed, if $m^e < n$, then from $c \equiv m^e \bmod n$ one obtains $m = \sqrt[e]{c}$.
    Prevention: salting.

RSA security (cont.)

- Small decryption exponent d
  - to increase the decryption speed: d might be small

    However, if $(p-1, q-1)$ is small, as it is typically the case, and if d has up to approximately one-quarter as many bits as the modulus n, then there is an efficient algorithm to compute d from the public information (n, e). The algorithm cannot be extended if d has approximately the same size as n.
    Prevention: choose d of roughly the same size as n.
- Small modulus n
  - a small modulus allows its factorization, ruining the security
- Forward search attack
  - the message should not be small or predictable

    Otherwise, an adversary can decrypt a ciphertext c by simply encrypting all possible plaintext messages until c is obtained.
    Prevention: salting.

RSA security (cont.)

- Multiplicative properties

  Let m_1, m_2 be plaintext messages and let c_1, c_2 be their corresponding ciphertext messages. Then

  $$(m_1 m_2)^e \equiv m_1^e m_2^e \equiv c_1 c_2 \pmod{n}.$$

  - This leads to the following *adaptive chosen-ciphertext attack*. Suppose that an adversary Eve wants to decrypt a ciphertext $c \equiv m^e \bmod n$ intended for Alice. Suppose also that Alice will decrypt arbitrary ciphertext for Eve, other than c itself. Then Eve can conceal c by selecting a random invertible integer $x \in \mathbb{Z}_n^*$ and computing $c' = cx^e \bmod n$. Upon presentation of c', Alice computes for Eve $m' = (c')^d \bmod n$. Since

    $$m' \equiv (c')^d \equiv c^d (x^e)^d \equiv mx \pmod{n},$$

    Eve can obtain $m = m'x^{-1} \bmod n$.

  Prevention: impose some structural constraints on plaintext messages. If a ciphertext is decrypted to a message not possessing this structure, then it is rejected.

RSA security (cont.)

- Common modulus attack
  Suppose that a central trusted authority selects a single modulus n and then distributes a distinct encryption / decryption exponent pair (e_i, d_i) to each entity in a network.
  - Knowledge of any (e_i, d_i) pair allows for the factorization of the modulus n, and hence any entity could subsequently determine the decryption exponents of all other entities in the network.
  - Let m be a message which has been encrypted with the public keys (n, e_1) and (n, e_2), hence

$$c_1 \equiv m^{e_1} \bmod n; \qquad c_2 \equiv m^{e_2} \bmod n.$$

    Since $\gcd(e_1, e_2) = 1$, $\exists a, b \in \mathbb{Z}$ with $1 = a \cdot e_1 + b \cdot e_2$. Hence

$$m \bmod n \equiv m^1 \bmod n \equiv (m^{e_1})^a \cdot (m^{e_2})^b \bmod n = c_1^a \cdot c_2^b \bmod n.$$

  Prevention: each user should have a different modulus n.

RSA security (cont.)

- Blinding

  Suppose an adversary Oscar wants Bob's signature on a message $m \in \mathbb{Z}_n^*$. Surely, Bob refuses to sign m for Oscar.

  But Oscar may pick a random number $r \in \mathbb{Z}_n^*$, set

  $$m' = r^e m \bmod n,$$

  and ask Bob to sign the random message m'.

  If Bob signs it with the signature s', then $s' = m'^d \bmod n$, and Oscar may compute the signature s for the original message m as

  $$s = s' r^{-1} \bmod n.$$

  One can check that:

  $$s^e = s'^e r^{-e} = m'^{ed} r^{-e} = m' r^{-e} = m \bmod n.$$

  Prevention: Most signature schemes apply a one-way hash to the message before signing, hence the attack is not a concern.

RSA security (cont.)

- Quantum cryptography: polynomial-time Shor's algorithm for factoring large integers.

- Timing attacks

  Consider a smartcard that stores a private RSA key.

  Kocher's attack discovers the private decryption exponent d by precisely measuring the time it takes the smartcard to perform an RSA decryption (or signature).

  Oscar asks the smartcard to generate signatures on a large number of random messages $m_1, \ldots, m_k \in \mathbb{Z}_n^*$ and measures the time it takes the card to generate each of the signatures.

  The algorithm uses repeated squaring modular exponentiation.

  Prevention: Add appropriate delay so that modular exponentiation always takes a fixed amount of time, or use blinding.

The first rule: we do not compute $C := M^e \pmod{n}$ by first exponentiating M^e and then performing a division to obtain the remainder.

If M and e have 256 bits each, then one needs

$$\log_2(M^e) = e \cdot \log_2(M) \approx 2^{256} \cdot 256 = 2^{264} \approx 10^{80}$$

bits in order to store M^e. This number is approximately equal to the number of particles in the universe.

The binary method scans the bits of the exponent. A squaring is performed at each step, and depending on the scanned bit value, a subsequent multiplication is performed.

Let k be the number of bits of e, and the binary expansion of e be given by $e = (e_{k-1}e_{k-2}\ldots e_1 e_0) = \sum_{i=0}^{k-1} e_i 2^i$ with $e_i \in \{0, 1\}$.

The Binary Method
Input: M, e, n.
Output: $C = M^e \bmod n$.
1. **if** $e_{k-1} = 1$ **then** $C := M$ **else** $C := 1$
2. **for** $i = k - 2$ **downto** 0
  2a. $C := C \cdot C \pmod{n}$
  2b. **if** $e_i = 1$ **then** $C := C \cdot M \pmod{n}$
3. **return** C

The Binary Method - cont.

For example, let $e = 250 = (11111010)$, which implies $k = 8$.
Initially, we take $C := M$, since $e_{k-1} = e_7 = 1$.

i	e_i	Step 2a	Step 2b
6	1	$(M)^2 = M^2$	$M^2 \cdot M = M^3$
5	1	$(M^3)^2 = M^6$	$M^6 \cdot M = M^7$
4	1	$(M^7)^2 = M^{14}$	$M^{14} \cdot M = M^{15}$
3	1	$(M^{15})^2 = M^{30}$	$M^{30} \cdot M = M^{31}$
2	0	$(M^{31})^2 = M^{62}$	M^{62}
1	1	$(M^{62})^2 = M^{124}$	$M^{124} \cdot M = M^{125}$
0	0	$(M^{125})^2 = M^{250}$	M^{250}

The number of modular multiplications is $7 + 5 = 12$.
For an arbitrary k-bit number e, the binary method requires:

- $k - 1$ squarings in Step 2a
- $H(e) - 1$ multiplications in Step 2b, where $H(e)$ is the Hamming weight (the number of 1's in the binary expansion) of e.
- $\frac{3}{2}(k - 1)$ average total number of multiplications.

The m-ary Method

The binary method can be generalized by scanning the bits of e:

- 2 at a time: the quaternary method
- 3 at a time: the octal method
- More generally, $\log_2(m)$ at a time: the m-ary method.

Let $e = (e_{k-1}e_{k-2}\ldots e_1 e_0)$ be the binary expansion of e.

This representation of e is partitioned into s blocks of length r each for $sr = k$. If r does not divide k, the exponent is padded with at most $r-1$ 0's. Define

$$F_i = (e_{ir+r-1}e_{ir+r-2}\ldots e_{ir}) = \sum_{j=0}^{r-1} e_{ir+j}2^j.$$

Note that $0 \leq F_i \leq m-1$ and $e = \sum_{i=0}^{s-1} F_i 2^{ir}$.

The m-ary Method - cont.

The m-ary method first computes the values $M^w \pmod{n}$ for $w = 2, 3, \ldots, m-1$.

Then the bits of e are scanned r bits at a time from the most significant to the least significant.

At each step the partial result is raised to the 2^r power and multiplied by M^{F_i} mod n, where F_i is the (nonzero) value of the current bit section.

The m-ary Method
Input: M, e, n.
Output: $C = M^e \bmod n$.

1. Compute and store $M^w \pmod{n}$ for all $w = 2, 3, 4, \ldots, m-1$.
2. Decompose e into r-bit words F_i for $i = 0, 1, 2, \ldots, s-1$.
3. $C := M^{F_{s-1}} \pmod{n}$
4. **for** $i = s-2$ **downto** 0
  4a. $C := C^{2^r} \pmod{n}$
  4b. **if** $F_i \neq 0$ **then** $C := C \cdot M^{F_i} \pmod{n}$
5. **return** C

The Quaternary Method

Since the bits of e are scanned two at a time, the possible digit values are $(00) = 0$, $(01) = 1$, $(10) = 2$, and $(11) = 3$.

The multiplication step (Step 4b) may require M^0, M^1, M^2, M^3.

We need to perform some preprocessing to obtain M^2, M^3.

For example, let $e = 250$ and partition the bits of e in groups of two bits as $e = 250 = 11\ 11\ 10\ 10$. Here, we have $s = 4$ (the number of groups $s = k/r = 8/2 = 4$).

The quaternary method assigns $C := M^{F_3} = M^3 \pmod{n}$, and proceeds to compute $M^{250} \pmod{n}$ as follows:

bits	w	M^w
00	0	1
01	1	M
10	2	$M \cdot M = M^2$
11	3	$M^2 \cdot M = M^3$

i	F_i	Step 4a	Step 4b
2	11	$(M^3)^4 = M^{12}$	$M^{12} \cdot M^3 = M^{15}$
1	10	$(M^{15})^4 = M^{60}$	$M^{60} \cdot M^2 = M^{62}$
0	10	$(M^{62})^4 = M^{248}$	$M^{248} \cdot M^2 = M^{250}$

The number of modular multiplications required by the quaternary method for computing $M^{250} \pmod{n}$ is $2 + 6 + 3 = 11$.

The Octal Method

This partitions the bits of the exponent in groups of 3 bits.
For example, $e = 250 = 011\ 111\ 010$, by padding a zero to the left, giving $s = k/r = 9/3 = 3$.
We precompute $M^w \pmod{n}$ for all $w = 2, \ldots, 7$.
The octal method then assigns $C := M^{F_2} = M^3 \pmod{n}$, and proceeds to compute $M^{250} \pmod{n}$ as follows:

bits	w	M^w
000	0	1
001	1	M
010	2	$M \cdot M = M^2$
011	3	$M^2 \cdot M = M^3$
100	4	$M^3 \cdot M = M^4$
101	5	$M^4 \cdot M = M^5$
110	6	$M^5 \cdot M = M^6$
111	7	$M^6 \cdot M = M^7$

i	F_i	Step 4a	Step 4b
1	111	$(M^3)^8 = M^{24}$	$M^{24} \cdot M^7 = M^{31}$
0	010	$(M^{31})^8 = M^{248}$	$M^{248} \cdot M^2 = M^{250}$

The computation of $M^{250} \pmod{n}$ by the octal method requires a total of $6 + 6 + 2 = 14$ modular multiplications.

The Octal Method - cont.

Notice that we have not used all $M^w \pmod{n}$ for $w = 2, \ldots, 7$. Thus, we can precompute $M^w \pmod{n}$ only for those w which appear in the partitioned binary expansion of e.

For example, for $e = 250$, the partitioned bit values are:
$(011) = 3, (111) = 7, (010) = 2$.
We can compute these powers using only 4 multiplications:

bits	w	M^w
000	0	1
001	1	M
010	2	$M \cdot M = M^2$
011	3	$M^2 \cdot M = M^3$
100	4	$M^3 \cdot M = M^4$
111	7	$M^4 \cdot M^3 = M^7$

Now the total number of multiplications required by the octal method for computing $M^{250} \pmod{n}$ is $4+6+2 = 12$.

The m-ary Method - conclusion

We summarize the average number of multiplications and squarings required by the m-ary method assuming that $2^r = m$ and $\frac{k}{r}$ is an integer.

- preprocessing multiplications (Step 1): $m - 2 = 2^r - 2$
- squarings (Step 4a): $(\frac{k}{r} - 1) \cdot r = k - r$
- multiplications (Step 4b): $(\frac{k}{r} - 1)(1 - \frac{1}{m}) = (\frac{k}{r} - 1)(1 - \frac{1}{2^r})$
- average total number of multiplications plus squarings:

$$(2^r - 2) + (k - r) + \left(\frac{k}{r} - 1\right)\left(1 - \frac{1}{2^r}\right)$$

The average number of multiplications for the binary method can be found simply by substituting $r = 1$ and $m = 2$ in the above, which gives $\frac{3}{2}(k - 1)$.

Selective Bibliography

- M. Cozzens, S.J. Miller, *The Mathematics of Encryption: An Elementary Introduction*, American Mathematical Society, 2013.
- Ç.K. Koç, *High-Speed RSA Implementation*, RSA Laboratories. [ftp://ftp.rsasecurity.com/pub/pdfs/tr201.pdf]
- A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997. [http://www.cacr.math.uwaterloo.ca/hac]