Introduction to Applied Cryptography, Part 2*

Prof. Susanne Wetzel
Stevens Institute of Technolgy
Department of Computer Science
Hoboken, New Jersey, USA

Dr. Torsten Schütze
Robert Bosch GmbH
Corporate Sector Research
and Advance Engineering
Software (CR/AEA)

September 21–23, 2009

*The material has been presented at previous Summer Schools for Studienstiftung des Deutschen Volkes while the second author was at Siemens AG, CT IC 3.

Outline of Part 2

1. RSA

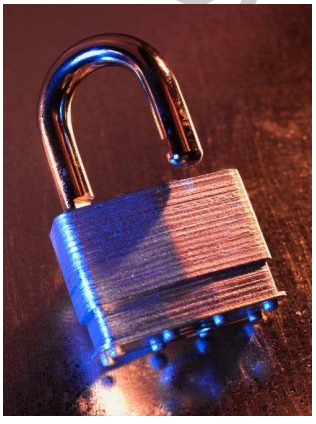
2. ElGamal / Discrete Logarithms / DSA

Part 2: Outline — Section 1: RSA

1. Public-Key Encryption

Definition 1.1.

An encryption scheme with encryption transformations $\{E_e : e \in \mathcal{K}\}$, decryption transformations $\{D_d : d \in \mathcal{K}\}$ is said to be *public key*, if for each associated encryption/decryption key pair (e, d) with $e, d \in \mathcal{K}$ one key e (public key) is made publicly available, while the other d (private key) is kept secret. For the scheme to be secure it must be computationally infeasible to compute d from e.



Public-Key Encryption (2)

- The encryption transformation should be injective, easy to compute, but hard to invert = injective one-way function
- Injective one-way function + easy to invert with knowledge of certain trapdoor = trapdoor one-way function
- Existence of one-way function (provably secure in a complexity theoretic sense) is open problem

Example 1.2. (believed) one-way function

$$f : \mathbb{Z}_n \to \mathbb{Z}_n \qquad n = pq, \text{ with } p, q \text{ prime}$$
$$f(x) := x^b \bmod n$$

If $\gcd(b, \phi(n)) = 1$, then f is the RSA function, see later.

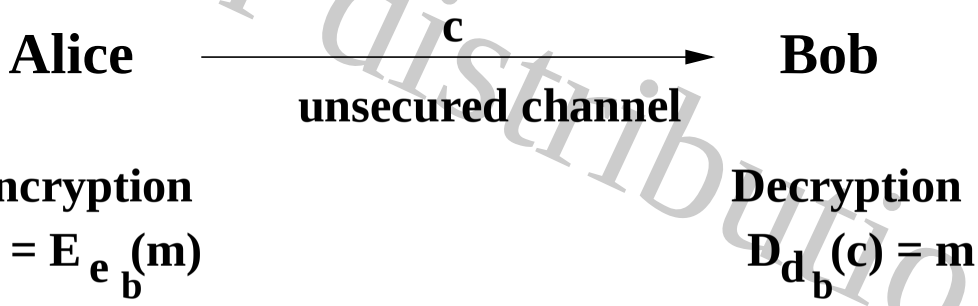
History of Public Key Cryptography

- James Ellis, 1970, Clifford Cocks, 1973, British GCHQ.
  idea and implementation of public key cryptography, revealed 1997!
- Whitfield Diffie, Martin Hellman: *New Directions in Cryptography*, 1976.
  idea of public key cryptography (public world), no concrete instantiation of trap-door one-way function
- Whitfield Diffie, Martin Hellman, (Ralph Merkle): discrete logarithm based key agreement
- Ron Rivest, Adi Shamir, Leonard Adleman: *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, 1977.
  RSA-method for signature and encryption.
- Shafi Goldwasser, Silvio Micali, Manual Blum, 1982.
  provable security = firm foundation of public key cryptography
- Neal Koblitz, Victor Miller, 1985.
  invention of Elliptic Curve Cryptography (signature, encryption, key agreement)
- 2005: NSA adopts Suite B (includes ECC package from Certicom)
- ECC for Bonn-Berlin Verbund, NATO encryption device, ICAO travel documents, german passports, etc.

Necessity of Authentication (1)

Want:

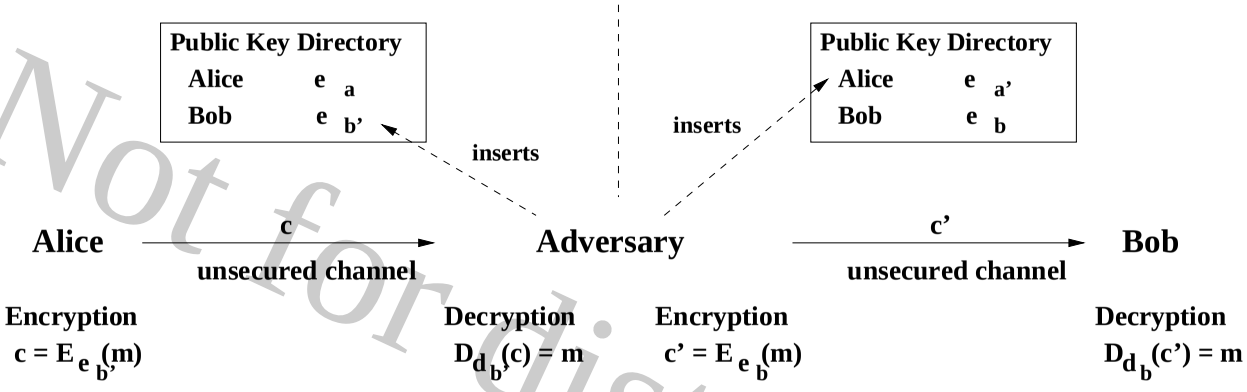
Public Key Directory	
Alice	e_a
Bob	e_b



e_a, e_b public encryption keys of Alice and Bob, d_b private decryption key of Bob

Necessity of Authentication (2)

Get:



Man-in-the-Middle-Attack ⇒ necessity to authenticate public keys

Symmetric Key vs. Public-Key Cryptography (1)

- Advantages of symmetric key cryptography:
  - Symmetric key ciphers have high rates of data throughput.
  - Keys for symmetric key ciphers are relatively short.
  - Symmetric key ciphers can be employed as primitives to construct various cryptographic mechanisms.
  - Symmetric key ciphers can be combined to produce stronger ciphers.
- Disadvantages of symmetric key cryptography:
  - Key management, i.e.,:
    - Number of keys
    - Key storage
    - Key exchange
  - Unconditionally trusted TTP (Trusted Third Party) is required.
  - Keys have to be changed frequently.

Symmetric Key vs. Public-Key Cryptography (2)

- Advantages of public-key cryptography:
  - No secret keys have to be transmitted, only private key must be kept secret.
  - Functionally trusted TTP is sufficient.
  - Key pairs may remain unchanged for considerable period of time.
  - Small number of keys.
- Disadvantages of public-key cryptography:
  - Throughput rates are several orders of magnitude slower.
  - Authenticity of public keys must be guaranteed (Public Key Infrastructure).
  - Key sizes are typically much larger.
  - More structure $\Longrightarrow$ (implementation) failures have more consequences
  - It is, in general, not recommended to encrypt large portions of structured text by asymmetric methods!

The Best of Both Worlds

Hybrid method:

- Use public-key encryption for transport of keys.
- Use symmetric algorithms for bulk data encryption.
- Widely deployed: simple combination of known schemes for
  - asymmetric encryption (symmetric key)
  - symmetric encryption (bulk data)
  - digital signatures (includes hash)

  in standards like OpenPGP and S/MIME
- State-of-the-Art in Cryptography: provably secure hybrid methods
  - ECIES (Elliptic Curve Integrated Encryption Scheme) = ECC encryption + AES symmetric encryption + hash = authenticated encryption
  - ISO 18033-2-2006. Information technology – Security techniques – Encryption Algorithms – Asymmetric Ciphers = generic hybrid ciphers based on ElGamal encryption and RSA transform

Useful Facts about Groups

- Lagrange theorem: Suppose G is a multiplicative group of order n and $g \in G$ is of order m (i.e., m is the smallest positive integer such that $g^m = 1$). Then m divides n.
- Euler: If $b \in \mathbb{Z}_n^*$, then $b^{\phi(n)} \equiv 1 \pmod{n}$ where $\phi(n) = |\mathbb{Z}_n^*|$.
- Fermat: Suppose p is prime and $b \in \mathbb{Z}_p$. Then $b^p \equiv b \pmod{p}$.
- If p is prime, then $\mathbb{Z}_p^*$ is a cyclic group, i.e., there exists an element $g \in \mathbb{Z}_p^*$ having order equal to $p-1$. g is called a primitive element.
- Suppose that p is prime and $g \in \mathbb{Z}_p^*$. Then g is a primitive element iff $g^{(p-1)/q} \not\equiv 1 \pmod{p}$ for all primes q such that $q|(p-1)$.

2. RSA — Key Generation

Each entity A should do the following:

1. Generate two large random, distinct primes p and q, each roughly the same size.
2. Compute the modulus $n = pq$ and $\phi = (p-1)(q-1)$.
3. Select a random integer e (encryption exponent) $1 < e < \phi$ s.t. $\gcd(e, \phi) = 1$.
4. Compute the unique integer d (decryption exponent) $1 < d < \phi$ s.t. $ed \equiv 1 \pmod{\phi}$.
5. A's public key is (n, e), the private key is d.

RSA — Key Generation in Practice

- There are certain attacks (Coppersmith, May etc.) for small private key d, e.g., $d < n^{0.292}$. If e is selected first, then d is large. (RSA1977 starts with d!)
- BSI requires for the generation of RSA primes (qualified digital signatures)
  - physical RNG with mechanism strength P2 high (AIS 31)
  - probabilistic prime number test with upper bound 2^{-100}
  - $\epsilon_1 < |\log_2 p - \log_2 q| < \epsilon_2$ with $\epsilon_1 \approx 0.1$, $\epsilon_2 \approx 30$
- $ed \equiv 1 \pmod{(p-1)(q-1)}$ was changed in PKCS#1 v2.0, 1998 to $ed \equiv 1 \pmod{\operatorname{lcm}(p-1, q-1)}$

period	–2008	–2009	–2010	–2015
n (required)	1280	1536	1728	1976
n (recommended)	2048	2048	2048	2048

Recommended RSA Key Length for Qualified Digital Signatures, BSI 2009.

RSA — Encryption/Decryption

Encryption: B who encrypts a message m for entity A should do the following:

1. Obtain A's authentic public key (n, e).
2. For the message m with $0 \leq m < n$ compute $c = m^e \bmod n$.
3. Send the ciphertext c to A.

Decryption: To recover the plaintext m from the received ciphertext c, A should do the following:

1. Use the private key d to recover $m = c^d \bmod n$.

RSA — Why does it work?

Theorem 1.3.
Let (n, e) be a public RSA key and d the corresponding private RSA key. Then

$$(m^e)^d \bmod n = m$$

for any integer m with $0 \leq m < n$.

Proof 1.4.
Since $ed \equiv 1 \bmod (p-1)(q-1)$, $\exists l \in \mathbb{Z}$ with $ed = 1 + l(p-1)(q-1)$. Thus, $(m^e)^d = m^{ed} = m^{1+l(p-1)(q-1)} = m(m^{(p-1)(q-1)})^l$. If $\gcd(m, p) = 1$, then $m^{(p-1)} \equiv 1 \bmod p$ (Fermat's theorem) and therefore

$$m(m^{(p-1)(q-1)})^l \equiv m \bmod p.$$

If $\gcd(m, p) = p$, then both sides are $0 \bmod p$. Analogously, $(m^e)^d \equiv m \bmod q$.

$$\Rightarrow (m^e)^d \equiv m \bmod n \qquad \square$$

Security of RSA

Theorem 1.5.

The problem of computing the RSA decryption exponent d from the public key (n, e) and the problem of factoring n are computationally equivalent.

Proof 1.6.

- Factoring of n known $\Rightarrow$ computation of ϕ and d as in key generation process.
- d known (Sketch):
  - Because of $ed \equiv 1 \bmod \phi$, $\exists k \in \mathbb{Z}$ with $ed = 1 + k\phi$. Hence, $a^{ed-1} \equiv 1 \bmod n \; \forall a \in \mathbb{Z}_n^*$ (Euler's theorem). Let $ed - 1 = 2^s t$, where t is an odd integer.
  - It can be shown that $\exists i$ in $[1, s]$ s.t. $a^{2^i t} \equiv 1 \bmod n$ and $a^{2^{i-1} t} \not\equiv \pm 1 \bmod n$ for at least half of all $a \in \mathbb{Z}_n^*$. If a is such an integer, then $\gcd(n, a^{2^{i-1} t} - 1)$ is a non-trivial factor of n. = randomized Las Vegas type algorithm, r runs, probability of factor found $\geq 1 - 2^{-r}$

Remark 1.7.

It is not known whether breaking RSA is as difficult as factoring integers.

RSA — Choice of e

- $e = 2$ not possible in RSA, but see Rabin method (1979): $c = m^2 \bmod n$
- Small e makes encryption efficient. Smallest possible: $e = 3$.
- Risk: low exponent attack!
  - Assume A sends $c_i = m^3 \bmod n_i$ for pairwise relatively prime moduli n_1, n_2, n_3. Observing the c_i's and using the Chinese Remainder theorem, the adversary can compute $m^3 < n_1 n_2 n_3$.
  - Attack can be prevented by salting.
- Often used: $e = F_4 = 2^{16} + 1 = 65537$, i. e., 17 multiplications for exponentiation with square and multiply.
- There are certain implementation attacks which work only for $e = 3$. Public exponent $e = 3$ is not recommended!
- Cryptographers nowadays recommend at least 32 bit for e, i.e., approx. 48 multiplications.

RSA — Multiplicative Property

Let m_1 and m_2 be two plaintext messages, and let c_1 and c_2 be their respective RSA encryptions. Thus,

$$(m_1 m_2)^e \equiv m_1^e m_2^e \equiv c_1 c_2 \bmod n.$$

$$\Rightarrow \text{homomorphic property}$$

Adaptive chosen ciphertext attack:

- Adversary wants to decrypt $c = m^e \bmod n$, intended for A.
- A is willing to decrypt arbitrary ciphertexts $c' \neq c$.
- Adversary selects $x \in \mathbb{Z}_n^*$ and computes $c' = cx^e \bmod n$.
- A will compute $m' \equiv c^d (x^e)^d \equiv mx \bmod n$.
- Attack can be prevented by imposing structural constraints on plaintext or apply padding.

RSA — Cycling Attack

Let $c = m^e \bmod n$ and k be an integer such that $c^{e^k} \equiv c \bmod n$. Then,

$$c^{e^{k-1}} \equiv m \bmod n.$$

Cycling attack:

- Compute $c^e \bmod n$, $c^{e^2} \bmod n$, ..., until c is obtained for the first time.

RSA — Common Modulus Attack

- Central trusted authority selects a single RSA modulus and distributes distinct pairs (e_i, d_i) to the participants.
- Knowledge of a single pair (e_i, d_i) allows factorization of n.
- Thus, any participant can subsequently determine the decryption exponents of all the others.

Math refresher: Chinese Remainder Theorem

Theorem 1.8. Chinese Remainder Theorem

Suppose $m_1, \ldots, m_r$ are pairwise relatively prime positive integers, and suppose $a_1, \ldots, a_r$ are integers. Then the system of r congruences $x \equiv a_i \pmod{m_i}$, $i = 1, \ldots, r$ has a unique solution modulo $M = m_1 \times \cdots \times m_r$, which is given by

$$x = \sum_{i=1}^{r} a_i M_i y_i \bmod M,$$

where $M_i = {}^{M}/_{m_i}$ and $y_i = M_i^{-1} \bmod m_i$, for $i = 1, \ldots, r$.

RSA — Efficiency

Use Chinese Remainder theorem to speed up decryption:

- To decrypt ciphertext c using private exponent d, compute $m_p = c^{d \bmod (p-1)} \bmod p$ and $m_q = c^{d \bmod (q-1)} \bmod q$.
- Find integer y_p and y_q, such that $y_p p + y_q q = 1$, e.g., with Euclidean Algorithm.
- Then,

$$m = (m_p y_q q + m_q y_p p) \bmod n.$$

- Note, $y_p p$ and $y_q q$ can be precomputed.
- The classical Gauss algorithm [1801] for CRT requires a modular reduction modulus n, thus, it is more expensive than the algorithm by Garner [1959], see next slide.

RSA — Efficiency: CRT in Practice

Algorithm Garner's Algorithm for RSA decryption with CRT

Input: $c, p, q, d_p = d \bmod (p-1), d_q = d \bmod (q-1), P_p = q^{-1} \bmod p$

$m_p := c^{d_p} \bmod p$

$m_q := c^{d_q} \bmod q$

$m := m_q + [(m_p - m_q) \times P_p \bmod p] \times q$

Output: $m := c^d \bmod n$

Remark 1.9.

- Exponent d is of order n, $d_p := d \bmod (p-1)$ is of order $p \approx \frac{n}{2}$!
- In theory, RSA-CRT is four times faster than direct exponentiation, in practice, often an acceleration about factor three can be achieved.
- The precomputed numbers $p, q, d_p, d_q, P_p = q^{-1} \bmod p$ are stored in a private key certificate format.
- CRT acceleration can only be used for RSA decryption and RSA signature generation. It is very vulnerable to implementation attacks, especially faults.

PKCS Overview

Many methods for Public Key Cryptography are standardized by RSA Security (now part of EMC Corp.) in Public Key Cryptography Standards, PKCS.

Standard	Title	Comment
PKCS#1	RSA Cryptography Standard	encryption, signature
PKCS#3	Diffie-Hellman Key Agreement Standard	
PKCS#5	Password-Based Cryptography Standard	key derivation
PKCS#6	Extended-Certificate Syntax Standard	
PKCS#7	Cryptographic Message Syntax Standard	
PKCS#8	Private-Key Information Syntax Standard	
PKCS#9	Selected Attribute Types	
PKCS#10	Certification Request Syntax Standard	
PKCS#11	Cryptographic Token Interface Standard	Hardware abstraction
PKCS#12	Personal Information Exchange Syntax Standard	
PKCS#13	Elliptic Curve Cryptography Standard	in development
PKCS#15	Cryptographic Token Information Format Standard	

Source: RSA Labs: `http://www.rsa.com/rsalabs/node.asp?id=2124`

RSA-Encryption in Practice — PKCS#1 Standard

- PKCS#1 is the relevant standard for RSA. It covers encryption and signatures.
  v1.5, 1993: current industry standard
  v2.0, 1998: quick fix due to attacks
  v2.1, 2002: new provably secure schemes, not yet widely deployed
- PKCS#1 v2.1 Encryption methods
  - `RSAES-PKCS1-v1_5`: RSA Encryption Scheme – PKCS#1 v1.5 padding
  - `RSAES-OAEP`: RSA Encryption Scheme – Optimal Asymmetric Encryption Padding
- PKCS#1 v2.1 Signature methods
  - `RSASSA-PKCS1-v1_5`: RSA Signature Scheme with Appendix – PKCS#1 v1.5 padding
  - `RSASSA-PSS`: RSA Signature Scheme with Appendix – Provably Secure Encoding Method for Digital Signatures
- Details about PKCS#1 v2.1 schemes, see later (RSA signatures)

RSA in Practice — Security of PKCS#1 v1.5 and v2.1

- `RSAES-PKCS1-v1_5` is believed to be CPA secure (no proof)
- `RSAES-PKCS1-v1_5` is not secure against adaptive Chosen-Ciphertext Attacks (Bleichenbacher, CRYPTO 1998, 1 mio calls of padding oracle to break OpenSSL)
  $\Longrightarrow$ ad hoc implementation countermeasures to fix v1.5
- Further attacks on v1.5 encryption by Klima/Pokorny/Rosa 2003 and other
- Attack on first version of `RSAES-OAEP` in PKCS#1 v2.0 by Manger, 2001
- `RSAES-OAEP` is CCA2-secure (Fujisaki, Okamoto, Pointcheval, Stern, 2001)
- CRYPTO 2006 Rump Session, Bleichenbacher: Existential Forgery Attack on `RSASSA-PKCS1-v1_5` implementations with $e = 3$
- PKCS#1 v1.5 implementations offer lower security than v2.1 and have to be carefully checked against implementation flaws!

> „Die Verwendung des PKCS#1 v1.5 Padding-Verfahrens für RSA-Signaturen soll, wie mehrfach bei den vergangenen Expertenanhörungen erwähnt, mittelfristig auslaufen. Das BSI hält das Verfahren bis Ende 2012 für geeignet (vgl. Entwurf), eine Eignung bis Ende 2013 erscheint fraglich."
>
> Quelle: Dr. G. Illies, Anschreiben des BSI zu geeigneten Signaturverfahren, 2006-09-05

3. RSA in Practice — Exponentiation Methods

Efficient computation of $c = M^e \bmod n$ with $e = (e_{k-1}, \ldots, e_0) = \sum_{i=0}^{k-1} e_i 2^i$ with $k = 1 + \lfloor \log_2 e \rfloor$ bitlength of e

---

Algorithm Left-to-right binary exponentiation, Square and Multiply

Input: M, e, n; **Output:** $c = M^e \bmod n$;
if $e_{k-1} = 1$ **then** $c := M$ **else** $c := 1$;
for $i := k - 2$ **downto** 0
    $c := c \times c \bmod n$; /* Square */
    if $e_i = 1$ **then** $c := c \times M \bmod n$; /* Multiply */
return c;

RSA in Practice — Exponentiation Methods (2)

Right-to-left binary exponentiation requires one additional variable.

Algorithm Right-to-left binary exponentiation

Input: M, e, n; **Output:** $c = M^e \bmod n$;

$c := 1; z := M;$

while $e \neq 0$ **do**

  if $e \bmod 2 = 1$ **then** $c := c \times z \bmod n$; /* Multiply */

  $e := e \operatorname{div} 2$;

  if $e \neq 0$ **then** $z := z \times z \bmod n$; /* Square */

return c;

Effort for Square and Multiply:

- Both binary methods have the same number of squarings and multiplications, but left-to-right uses fixed multiplicand M!
- $(k-1)$ squarings + $(H(e)-1)$ multiplications with $H(e)$ = Hamming weight

RSA in Practice — Exponentiation Methods (3)

Example 1.10. Signature Verification with Fermat number F_4

$e = 2^{16} + 1 = 65537 = (10000000000000001)$,

$k = 17 \Longrightarrow$ 16 squarings and 1 multiplication = 17 multiplications,

i.e., only modest increase in computation time compared with $e = 3$, no additional resources!

On average, i.e., random exponent, Square and Multiply requires $\frac{3}{2}(k-1)$ multiplications

Improvement over Square and Multiply: scan $2, 3, \ldots, r = \log_2 m$ bits of e at a time

$\Longrightarrow$ quaternary, octal,..., m-ary method

RSA in Practice — Exponentiation Methods (4)

binary representation of e is partitioned into s blocks of length r, $sr = k$

$$e = \sum_{i=0}^{s-1} F_i 2^{ir} \quad \text{with } 0 \leq F_i \leq m-1, F_i = (e_{ir+r-1} e_{ir+r-2} \cdots e_{ir}) = \sum_{j=0}^{r-1} e_{ir+j} 2^j$$

Algorithm m-ary left-to-right exponentiation

Precomputation: Compute and store $M^w \bmod n \qquad w = 2, 3, \ldots, m-1$

Decompose e into r-bit words F_i $i = 0, \ldots, s-1$

$c := M^{F_{s-1}} \bmod n$

for $i := s-2$ **downto** 0

  $c := c^{2^r} \bmod n$;

  if $F_i \neq 0$ **then** $c := c \times M^{F_i} \bmod n$;

Output: $c = M^e \bmod n$;

RSA in Practice — Exponentiation Methods (5)

Remark 1.11.

- For each k there exists an optimal window width r^* such that the average number of multiplications in the m-ary window method is minimum:

$$k = 512, 1024, 2048 \qquad \Longrightarrow \qquad r^* = 5, 5, 6$$

- m-ary methods require m additional variables for precomputation
  $\Longrightarrow$ large m not suitable for embedded systems
- Further improvements = adaptive m-ary methods, sliding windows

RSA in Practice — Multiple-Precision Multiplication

School method (pencil and paper algorithm)

To multiply two k-bit (s-digit) numbers, scan digits of multiplier and multiply digit with multiplicand to produce partial product, sum these partial products

$a = (a_{s-1}, a_{s-2}, \ldots, a_0) = \sum_{j=0}^{s-1} a_i W^i$, multiplicand
$b = (b_{s-1}, b_{s-2}, \ldots, b_0) = \sum_{j=0}^{s-1} b_i W^i$, multiplier
$a_i, b_i \in [0, W-1]$ digits, radix $W = 2^\omega$ with ω word size, e. g., $\omega = 32$

Example 1.12. School method for $s = 3$

				a_2	a_1	a_0
$\times$				b_2	b_1	b_0
				t_{02}	t_{01}	t_{00}
			t_{12}	t_{11}	t_{10}	
$+$		t_{22}	t_{21}	t_{20}		
	t_5	t_4	t_3	t_2	t_1	t_0

$t_{ij} = (\text{Carry, Sum})$ pair of $a_i b_j$ with radix W

RSA in Practice — Multiple-Precision Multiplication (2)

Algorithm School method for multiple-precision multiplication

Input: $a = (a_{s-1}, \ldots, a_0)$, $b = (b_{s-1}, \ldots, b_0)$ s-digit multiplicand and multiplier
Output: $t = ab = (t_{2s-1}, \ldots, t_0)$ $2s$-digit product

$t_i := 0; \quad i = 0, \ldots, 2s-1$
for $i := 0$ **to** $s-1$ **do**
  $C := 0$; /* carry */
  for $j := 0$ **to** $s-1$ **do**
    $(C, S) := t_{i+j} + a_i \times b_j + C$; /* inner product, row by row */
    $t_{i+j} := S$;
  $T_{i+s} := C$;
return $(t_{2s-1}, \ldots, t_0)$

Efficient implementation of inner product is most important task, required: s^2 steps with $s = {}^k/_\omega$ and ω computer constant

Effort for school method: $\mathcal{O}(k^2)$ operations to multiply two k-bit numbers

RSA in Practice — Fast Multiple-Precision Multiplication

Karatsuba-Ofman (1962) = asymptotically fast multiplication

$$\begin{aligned} ab &= (a_1 2^l + a_0)(b_1 2^l + b_0) \\ &= a_1 b_1 2^{2l} + [(a_0 + a_1)(b_0 + b_1) - a_1 b_1 - a_0 b_0]2^l + a_0 b_0 \end{aligned}$$

i.e. three instead of four multiplications, but more additions; Recursive application (divide and conquer) leads to complexity $\mathcal{O}(k^{\log_2 3}) = \mathcal{O}(k^{1.58})$

- Overhead for Karatsuba makes it only useful for very long integer multiplications, e.g., not for ECC with 256 bit, but for 2048-bit RSA in software.
- Threshold is about 320 bit on 32-bit computers depending very much on the arithmetic, see GMP configuration.
- In hardware, Karatsuba-Ofman leads to long delays. Thus, only hybrid forms are used (for GF(2^m)).
- Other fast multiple-precision multiplication methods as Toom Cook and FFT are used for very, very long operands, see GMP.

RSA in Practice — Montgomery Multiplication

Modular reduction $\bmod n$ is very expensive for general n
$\Longrightarrow$ Peter Montgomery (1985): replace division by n with division by a power of 2

$R = a \times b \bmod n$ $\quad$ a, b, n k-bit numbers

Assume $2^{k-1} \leq n < 2^k$ with $r = 2^k$.
Requirement: $\gcd(r, n) = \gcd(2^k, n) = 1$.
This requirement is satisfied for odd n, e. g., $n = pq$. It can be relaxed.

Definition 1.13. *n-residue* of $a < n$ with respect to r

$$\bar{a} := a \times r \bmod n$$

Note, that there is a one-to-one correspondence of the range $0, \ldots, n-1$ to the set $\{i \times r \bmod n \,|\, i = 0, \ldots, n-1\}$ (complete residue system).

RSA in Practice — Montgomery Multiplication (2)

Definition 1.14. Montgomery product of two n-residues $\bar{a}$ and $\bar{b}$

$$\bar{R} := \bar{a} \times \bar{b} \times r^{-1} \bmod n \qquad \text{with } r^{-1} \times r = 1 \bmod n.$$

Above number $\bar{R}$ is indeed the n-residue of $R = a \times b \bmod n$ since

$$\begin{aligned} \bar{R} &= \bar{a} \times \bar{b} \times r^{-1} \bmod n \\ &= a \times r \times b \times r \times r^{-1} \bmod n \\ &= a \times b \times r \bmod n \qquad \square \end{aligned}$$

Montgomery product algorithm produces $u = \bar{a} \times \bar{b} \times r^{-1} \bmod n$ only with multiplications modulo r and divisions by r (power of 2!)

RSA in Practice — Montgomery Multiplication (3)

Let n' be defined by $r \times r^{-1} - n \times n' = 1$ (can be computed by extended Euclidean algorithm, $\mathrm{xgcd}(r, n) = 1$), i. e., $n' = -n^{-1} \bmod r$.

Algorithm Montgomery Product of two n-residues: $\mathtt{MonPro}(\bar{a}, \bar{b})$

Input: $\bar{a} := a \times r \bmod n$, $\bar{b} := b \times r \bmod n$; **Output:** $u = \bar{a} \times \bar{b} \times r^{-1} \bmod n$

$t := \bar{a} \times \bar{b}$;
$m := t \times n' \bmod r$;
$u := (t + m \times n)/r$; /* $u := [\bar{a} \times \bar{b} + (\bar{a} \times \bar{b} \times n' \bmod r) \times n]/r$;*/
if $u \geq n$ **then return** $u - n$
else return u;

Montgomery Product Algorithm can often be found under Montgomery reduction for $zr^{-1} \bmod n$ with $z < nr$, see e. g., Hankerson/Vanstone/Menezes or Handbook of Applied Cryptography. Our presentation follows Koc (1994).

RSA in Practice — Montgomery Multiplication (4)

Algorithm Montgomery Modular Multiplication: $\texttt{MonMul}(a, b, n)$

Input: a, b, n with n odd, $2^{k-1} \leq n < 2^k$, $r = 2^k$; **Output:** $x = a \times b \bmod n$

Compute n' using extended Euclidean algorithm $\text{xgcd}(r, n) = 1$

$\bar{a} := a \times r \bmod n$; /* n-residue of a */

$\bar{b} := b \times r \bmod n$; /* n-residue of b */

$\bar{x} := \texttt{MonPro}(\bar{a}, \bar{b})$; /* Montgomery Product */

$x := \texttt{MonPro}(\bar{x}, 1)$; /* back transformation */

return x

It holds $\texttt{MonPro}(\bar{a}, b) = (a \times r) \times b \times r^{-1} \bmod n = ab \bmod n$

$\Longrightarrow$ improved Montgomery Modular multiplication

RSA in Practice — Montgomery Multiplication (5)

Algorithm Improved Montgomery Modular Multiplication: $\texttt{MonMul}(a, b, n)$

Input: a, b, n with n odd, $2^{k-1} \leq n < 2^k$, $r = 2^k$; **Output:** $x = a \times b \bmod n$

Compute n' using extended Euclidean algorithm $\text{xgcd}(r, n) = 1$

$\bar{a} := a \times r \bmod n$; /* n-residue of a */

$x := \texttt{MonPro}(\bar{a}, b)$; /* Montgomery Product */

return x

RSA in Practice — Montgomery Multiplication (5)

Precomputation overhead $\Longrightarrow$ Montgomery Modular Multiplication is not suited for single modular multiplications, only for a series of modular multiplications with constant modulus, i. e., modular exponentiation

---

Algorithm Left-to-right binary exponentiation using Montgomery Product

Input: m, e, n with n odd; **Output:** $m^e \bmod n$

Compute n' using extended Euclidean algorithm $\mathrm{xgcd}(r, n) = 1$

$\bar{m} := m \times r \bmod n$;

$\bar{x} := 1 \times r \bmod n$;

for $i := k - 1$ **downto** 0 **do**

    $\bar{x} := \mathtt{MonPro}(\bar{x}, \bar{x})$;

    if $e_i = 1$ **then** $\bar{x} := \mathtt{MonPro}(\bar{m}, \bar{x})$;

$x := \mathtt{MonPro}(\bar{x}, 1)$;

return x

RSA in Practice — Montgomery Multiplication (6)

Remark 1.15.

1. The Montgomery reduction step is suspectible to timing attacks.
2. Real world RSA implementation in OpenSSL:
   - 1024 bit RSA encryption
   - Garner's algorithm for CRT
   - Sliding window exponentiation with width 5
   - Montgomery multiplication as modular multiplication algorithm
   - Karatsuba acceleration
3. Attacks on this implementation: Schindler (2001), Boneh/Brumley (2003), see later Timing attacks

Definition 1.16.

The *integer factorization problem* is the following: given a positive integer n, find its prime factorization; that is, write $n = p_1^{e_1} \cdots p_k^{e_k}$ where the p_i are distinct primes and each $e_1 \geq 1$.

For a well-chosen RSA system, the factorization of the modulus n is still the most efficient attack to break RSA.

Name Number of Bits	Number Decimal places	Date	Participants
RSA-576	174	12/2003	Uni Bonn + BSI
RSA-200 (663)	200	05/2005	Uni Bonn + BSI
RSA-640	193	11/2005	Uni Bonn + CWI Amsterdam + BSI

Current Factorization Records

A factor of the Mersenne number $2^{1039} - 1$ with 307 decimal places has been factored in May 2007. This largest factored number has a very special structure.

Factoring — Trial Division

1. Precompute a table of all prime numbers below a fixed bound B with the sieve of Eratosthenes:
   (a) Write the list of integers $2, 3, 4, 5, \ldots, C$.
   (b) For $i = 2, 3, 4, 5, \ldots, C$ do:
      i. If i is still in the list, delete all proper multiples $2i, 3i, \ldots$ in the list.

   The numbers remaining in the list are the prime numbers $\leq C$.
2. For each prime p in the table, the maximum exponent $e(p)$ is determined s.t. $p^{e(p)} | n$.
3. A typical bound is $C = 10^6$

Factoring — Quadratic Sieve

Idea:

- The QS finds integers x and y such that $x^2 \equiv y^2 \bmod n$ and $x \not\equiv \pm y \bmod n$.
  $\Longrightarrow$ n divides $x^2 - y^2 = (x-y)(x+y)$, but not $x+y$ or $x-y$.
- Let $m = \lfloor \sqrt{n} \rfloor$ and

$$q(x) = (x+m)^2 - n = x^2 + 2mx + m^2 - n \approx x^2 + 2mx$$

which is small (relative to n if x is small in absolute value).

Factoring — Quadratic Sieve (2)

Example 1.17.

Let $n = 7429$. Then $m = 86$ and $q(x) = (x+86)^2 - 7429$. We have

$$q(-3) = 83^2 - 7429 = -540 = -1 * 2^2 * 3^3 * 5$$
$$q(1) = 87^2 - 7429 = 140 = 2^2 * 5 * 7$$
$$q(2) = 88^2 - 7429 = 315 = 3^2 * 5 * 7.$$

Thus,

$$83^2 \equiv -1 * 2^2 * 3^3 * 5 \bmod 7429$$
$$87^2 \equiv 2^2 * 5 * 7 \bmod 7429 \text{ and } 88^2 \equiv 3^2 * 5 * 7 \bmod 7429.$$

Therefore, $87^2 * 88^2 \equiv 2^2 * 3^2 * 5^2 * 7^2 \bmod n$ and $x = 87 * 88 \bmod n = 227$, $y = 2 * 3 * 5 * 7 \bmod n = 210$.
Hence, $x - y = 17$, $x + y = 437$, $\gcd(x-y, n) = \gcd(17, 7429) = 17$, i. e. factor 17 of n found.

Factoring — Quadratic Sieve (3)

- Select $a_i = (x + m)$ such that $b_i = (x + m)^2 - n$ has only small prime factors. Then use the fact that $b_i \bmod n \equiv a_i^2$.
- From congruences select subset whose product yields square on left and right. Product of a number of right-hand sides is a square if all exponents are even. = Sieving step
- Selection of congruences by solving a linear system: = Matrix step
  - Determine coefficients $\lambda_i \in \{0, 1\}$, $1 \leq i \leq 3$. Product of all right-hand sides is:

$$(-1 * 2^2 * 3^3 * 5)^{\lambda_1} * (2^2 * 5 * 7)^{\lambda_2} * (3^2 * 5 * 7)^{\lambda_3} =$$

$$(-1)^{\lambda_1} * 2^{2\lambda_1+2\lambda_2} * 3^{3\lambda_1+2\lambda_3} * 5^{\lambda_1+\lambda_2+\lambda_3} * 7^{\lambda_2+\lambda_3}.$$

  - Solve the following linear system: $\lambda_1 \equiv 0 \bmod 2$, $2\lambda_1 + 2\lambda_2 \equiv 0 \bmod 2$, $3\lambda_1 + 2\lambda_3 \equiv 0 \bmod 2$, $\lambda_1 + \lambda_2 + \lambda_3 \equiv 0 \bmod 2$, $\lambda_2 + \lambda_3 \equiv 0 \bmod 2$.
  - It simplifies to $\lambda_1 \equiv 0 \bmod 2$, $\lambda_1 + \lambda_2 + \lambda_3 \equiv 0 \bmod 2$, $\lambda_2 + \lambda_3 \equiv 0 \bmod 2$.
  - A solution is $\lambda_1 = 0$, $\lambda_2 = \lambda_3 = 1$.

Factoring — Quadratic Sieve (4)

- How to find the (a_i, b_i)?
  - Choose $x = 0, \pm 1, \pm 2, \pm 3, \ldots$
    Disadvantage: b_i typically not B-smooth, i.e., b_i has prime factors that do not belong to the factor base

$$S = \{p \text{ prime } : p \leq B\} \cup \{-1\}.$$

  - Sophisticated sieving methods.

Quadratic Sieve (5)

INPUT: a composite integer n that is not a prime power.
OUTPUT: a non-trivial factor d of n.

1. Select the factor base $S = \{p_1, p_2, \ldots, p_t\}$, where $p_1 = -1$ and p_j $(j \geq 2)$ is the $(j-1)^{\text{th}}$ prime p for which n is a quadratic residue modulo p.
2. Compute $m = \lfloor \sqrt{n} \rfloor$.
3. (*Collect $t+1$ pairs (a_i, b_i). The x values are chosen in the order $0, \pm 1, \pm 2, \ldots$.*)
   Set $i \leftarrow -1$. While $i \leq t+1$ do the following:
   3.1 Compute $b = q(x) = (x+m)^2 - n$, and test using trial division by elements in S whether b is p_t-smooth. If not, pick a new x and repeat step 3.1.
   3.2 If b is p_t-smooth, say $b = \prod_{j=1}^{t} p_j^{e_{ij}}$, then set $a_i \leftarrow (x+m)$, $b_i \leftarrow b$, and $v_i = (v_{i1}, v_{i2}, \ldots, v_{it})$, where $v_{ij} = e_{ij} \bmod 2$ for $1 \leq j \leq t$.
   3.3 $i \leftarrow i+1$.
4. Use linear algebra over $\mathbb{Z}_2$ to find a non-empty subset $T \subseteq \{1, 2, \ldots, t+1\}$ such that $\sum_{i \in T} v_i = 0$.
5. Compute $x = \prod_{i \in T} a_i \bmod n$.
6. For each j, $1 \leq j \leq t$, compute $l_j = (\sum_{i \in T} e_{ij})/2$.
7. Compute $y = \prod_{j=1}^{t} p_j^{l_j} \bmod n$.
8. If $x \equiv \pm y \pmod{n}$, then find another non-empty subset $T \subseteq \{1, 2, \ldots, t+1\}$ such that $\sum_{i \in T} v_i = 0$, and go to step 5. (In the unlikely case such a subset T does not exist, replace a few of the (a_i, b_i) pairs with new pairs (step 3), and go to step 4.)
9. Compute $d = \gcd(x-y, n)$ and return(d).

Source: Handbook of Cryptography

Run-times of algorithms

Consider a (probabilistic) algorithm with input of order n, i.e., bit length $\log_2 n$. Let the (expected) run-time be

$$L_n[\alpha, c] = \mathcal{O}\left(\exp\left((c + o(1))(\ln n)^{\alpha}(\ln \ln n)^{1-\alpha}\right)\right) \text{ with } c > 0 \text{ and } 0 \leq \alpha \leq 1.$$

- If $\alpha = 0$, polynomial run-time

$$L_n[0, c] = \mathcal{O}\left((\ln n)^{c+o(1)}\right).$$

- If $\alpha = 1$, exponential run-time

$$L_n[1, c] = \mathcal{O}\left(\exp\left((\ln n)^{c+o(1)}\right)\right).$$

- If $0 < \alpha < 1$, subexponential run-time.

Overview Factorization Methods

- Trial division, exponential run-time
- Pollard $p-1$ method (Pollard 1974), efficient if $p-1$ has small prime factors
  $\Longrightarrow$ use strong primes
- Elliptic Curve Method (Lenstra 1987), improvement of Pollard $p-1$ method, works for arbitrary n,
  subexponential run-time $L_p[1/2, \sqrt{1/2}]$, p smallest prime factor of n
- Quadratic Sieve (Pomerance 1981),
  subexponential run-time $L_n[1/2, 1+o(1)]$
- Number Field Sieve (Pollard 1988, Pomerance 1996),
  subexponential run-time $L_n[1/3, 1.92]$

The General Number Field Sieve is currently the most efficient method for integer factorization, it has subexponential run-time $L_n[1/3, 1.92]$.

Above results are for classical computers, in the quantum computer model there exists the polynomial factorization algorithm by Shor [1981]!

5. Digital Signatures

- Just as we used secret key cryptography to provide confidentiality (through encryption) and authentication (through MACs), public-key cryptography can also provide confidentiality (through encryption) and authentication (through digital signatures).
- **Idea:** Use private key to sign, public key to verify.
- Digital signatures also verify the originator of the message.
- Stronger property than with symmetric methods: Non-repudiation property: Only the owner of the private key can sign and can't later deny its signature!
- Analogous to handwritten signatures, but better, as signature is linked to the contents of the document is signed.

Signature Schemes

Formally, a signature scheme is a 5-tuple $(\mathcal{P}, \mathcal{A}, \mathcal{K}, \mathcal{S}, \mathcal{V})$, where

1. $\mathcal{P}$ is a finite set of possible messages.
2. $\mathcal{A}$ is a finite set of possible signatures.
3. $\mathcal{K}$, the keyspace, is a set of possible keys.
4. For each $K \in \mathcal{K}$, there is a signing algorithm $\text{sig}_K \in \mathcal{S}$ and a corresponding verification algorithm $\text{ver}_K \in \mathcal{V}$. Each $\text{sig}_K : \mathcal{P} \to \mathcal{A}$ and $\text{ver}_K : \mathcal{A} \to \{true, false\}$ are functions such that for every $m \in \mathcal{P}$ and $s \in \mathcal{A}$:

$$\text{ver}_K(m, s) = \begin{cases} true & \text{if } s = \text{sig}_K(m) \\ false & \text{if } s \neq \text{sig}_K(m) \end{cases}$$

The pair (m, s) is a signed message, and s is the signature of m.

Remark 1.18.

In public-key signature schemes, $K = (PK, SK)$, where in fact ver_K could be written ver_{PK} (and we will sometimes do so). That is, the verification algorithm can be computed from the public key alone.

Signature Schemes, cont'd

- The functions sig_K and ver_K should be efficiently computable. Given a message m, it should be infeasible for anyone other than the owner (say, Alice) of the public key to compute any signature s such that $\text{ver}_K(m, s) = true$.
- If an attacker who does not know Alice's private key can compute (m, s) such that $\text{ver}_K(m, s) = true$ and m was not previously signed by Alice, then (m, s) is a forgery.
- Note that in order that Alice's signature cannot be reused in unexpected ways, the message to be signed should be specific about its intended usage:

On November 15, 2004 at 5pm, Alice Jones borrowed $100 from Bob Smith.

vs.

I owe you $100.

Signatures from Reverse Encryption

- If encryption and decryption commute (i.e., $\text{decrypt}_{SK}(\text{encrypt}_{PK}(m)) = m$), then one might hope to be able to use decryption for signing and encryption for verifying. That is, Alice would create a signature using decryption:

$$s = \text{sig}_{SK_A}(m) = \text{decrypt}_{SK_A}(m)$$

and (m, s) would be the signed message. Bob would verify it using encryption:

$$\text{to compute } \text{ver}_{PK_A}(m, s), \text{ checks whether } \text{encrypt}_{PK_A}(s) \stackrel{?}{=} m$$

RSA Signature Scheme

- Has the usual RSA set-up: $n = pq$ where p and q are primes, $\mathcal{P} = \mathcal{A} = \mathbb{Z}_n$, and $ed \equiv 1 \mod \phi(n)$. Public (verification) key is n, e and private (signing) key is n, d.
- $\text{sig}_{n,d}(m) = m^d \mod n$
- $\text{ver}_{n,e}(m, s) = \textit{true}$ iff $m \equiv s^e \mod n$

Clearly, the verifier Bob considers the signature valid if s was correctly computed by the correct signer Alice.

6. Existential Forgery in RSA Signatures

BUT: anyone knowing Alice's public key can fool also Bob into considering a signature valid. An attacker could

- take any message t
- compute $x = t^e \bmod n$
- claim (x, t) as a signature

Remark 1.19.

Note that x is not known to the attacker, and may not be a meaningful message, but certainly (x, t) passes the signature verification procedure. This is called an existential forgery.

Types of Forgeries

- existential forgery: an adversary can create a valid signature for at least one message that has not been signed by Alice.
- selective forgery, also sometimes called universal forgery: an adversary can create a valid signature on a message chosen by someone else.
- total break: an adversary can determine Alice's private key

Remark 1.20.

All of the above should be read as "a randomized polynomial-time bounded adversary succeeds with non-negligible probability".

Types of Attack

As with encryption, the adversary may have access to the public key only, or some valid message/signature pairs, potentially to messages of its choosing.

- key-only attack: The adversary has only the public key.
- known message attack: The adversary has one or more valid message/signature pairs.
- chosen message attack: The adversary has one or more valid message/signature pairs, for messages of its choosing. In this case, the choice can be either required to be made all at once, or adaptively.

Remark 1.21.
Security is then described by resistance (or lack of resistance) to types of forgeries under various types of attack.

7. Hash and Sign Paradigm with Reverse Encryption

Idea: In order to avoid existential forgeries, instead of signing a message directly, a hash function is applied first. Since hashing is typically much faster than public-key operations, this also has the advantage that arbitrarily long messages can be signed with a single public-key operation.

- h is publicly known hash function. (It can be thought of as either part of the signature scheme or part of the public key).
- Alice creates a signature on m as follows:

$$s = \text{sig}_{SK_A}(m) = \text{decrypt}_{SK_A}(h(m))$$

  The pair (m, s) is still the signed message.
- To compute $\text{ver}_{PK_A}(m, s)$, Bob computes $h(m)$ and $\text{encrypt}_{PK_A}(s)$, and checks whether they are equal:

$$\text{ver}_{PK_A}(m, s) = \left(\text{encrypt}_{PK_A}(s) \stackrel{?}{=} h(m)\right)$$

Hash and Sign Paradigm

The intuition is that the security properties of the hash function will prevent an attacker from using a signature to create a forgery. (In contrast, in the reverse encryption scheme, the security of the encryption prevents an attacker from using a message to create a forgery.) We'll see a more precise argument of this intuition later. The hope is that:

- If h is preimage resistant, the signature scheme should resist existential forgery under a key-only attack.
- If h is second preimage resistant, the signature scheme should resist existential forgery under known message attacks.
- If h is collision resistant, the signature scheme should resist existential forgery under chosen message attacks.
- Recent attacks on hash functions MD5 and SHA-1: $\Longrightarrow$ no collision resistance $\Longrightarrow$ no non-repudiation property

Hash and Sign Paradigm, cont'd

Intuition behind why second preimage resistance is useful for preventing existential forgery under known message attacks:

> Given one signed message (m, s), one might try to find a forgery (m', s) by finding a second message m' with the same signature as m. But, in order for m and m' to have the same signature in the hash and sign paradigm, they must have the same hash output. That is, this would require finding an m' with $h(m') = h(m)$, which can't be done because h is second preimage resistant.

Remark 1.22.

Since *other* attacks besides the obvious ones could work, actually using hash functions for provably secure signature systems requires more involved constructions and/or the random oracle methodology.

8. State-of-the-Art: Qualified Digital Signatures

BSI (Bundesamt für Sicherheit in der Informationstechnik. Geeignete Kryptogalgorithmen. In Erfüllung nach §17(1) SigG vom 22. Mai 2001 in Verbindung mit Anlage 1, I2, SigV 22. November 2001. `http://www.bsi.de/esig/kryptoalg.htm`, Bundesanzeiger, 27. Januar 2009.

- Yearly catalog of suitable algorithms and parameters
- Signature schemes: RSA, DSA, EC(K,G)DSA (ECC over F_p and F_{2^m}!), ...
- Hash function: (SHA-1), (RIPEMD-160), SHA-224, SHA-256, SHA-384, SHA-512
- Random numbers: required: Pseudo-RNG at least K3 high, recommended: K4 high, key generation: recommended physical RNG P2 high, entropy of seed: 100 bit
- + additional conditions for parameters

period	–2008	–2009	–2010	–2015
n (required)	1280	1536	1728	1976
n (recommended)	2048	2048	2048	2048

Required RSA Key Length

State-of-the-Art: Qualified Digital Signatures (2)

period	–2008	–2009	–2015
p (required)	1280	1536	2048
p (recommended)	2048	2048	2048
q	160	160	224

Required DSA Key Length

period	–2009	–2015
p	192	
q	180	224

Required EC(K,G)DSA Key Length over F_p

period	–2009	–2015
m	191	
q	180	224

Required EC(K,G)DSA Key Length over F_{2^m}

9. RSA in Practice — Duality Encryption/Signature

- Formal duality of RSA encryption and signature formulae
  - signature generation for message m: $s = m^d \bmod n$,
    verification of signature s: $m = s^e \bmod n$
  - encryption of message m: $c = m^e \bmod n$,
    decryption of ciphertext c: $m = c^d \bmod n$
- BUT: never implement plain RSA!
  - Security notions for encryption and signature are different
  - Padding schemes are essential part of the RSA security system
  - Padding for encryption and signature is fundamentally different!

    > *. . . in signature verification, nondeterministic padding would be the worst thing to have since the whole point is that the recipient needs to be able to verify it.* Nate Lawson (Cryptographic Research), 2006-09-13

  - This is no contradiction to randomized signatures (DSA, ECDSA, PSS)!
- $\Longrightarrow$ RSA encryption and signature algorithms are not dual

RSA in Practice — PKCS#1 v1.5 Signature Generation

1. *Message hashing:* $h(M) \Longrightarrow$ octet string MD (`Message Digest`)
   Example: SHA-1, RIPEMD-160 $\Longrightarrow$ MD 20 Byte
2. *Message digest encoding:*
   `DigestAlgorithmIdentifier` + `Message Digest` = `DigestInfo`
   BER-encoded `DigestInfo` $\Longrightarrow$ octet data string D
   Example: D=3021300906052b0e03021a05000414||h(m) for SHA-1, D 35 Byte
3. *Data block formating:* BT=01 *Block Type* (BT=00 variant with PS=00), PS=FF. . . FF *Padding String*, octet string EB `Encryption Block`, length(EB)=length(n)=k
   EB = 00||BT||PS||00||D
   Example: RSA 1024 with SHA-1 (EB 128 Byte, 90 FF Byte):
   EB = 00||01||FF||. . . ||FF||00||3021300906052b0e03021a05000414||h(m)
4. *Octet string to integer conversion:* EB $\Longrightarrow$ integer m, $0 \leq m < n$
5. *RSA computation:* $s = m^d \bmod n$
6. *Integer to octet string conversion:* integer $s \Longrightarrow$ signature S

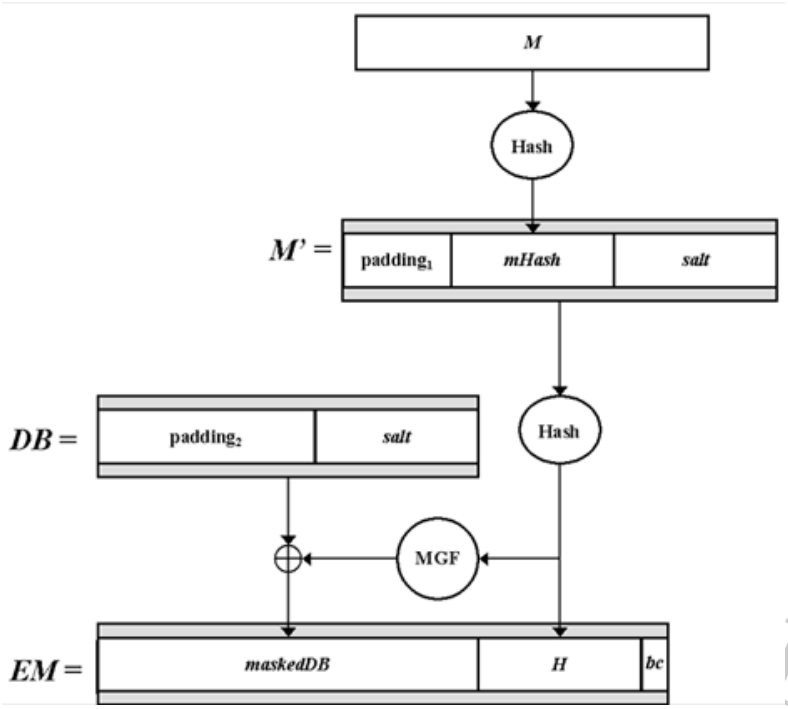
RSA in Practice — PKCS#1 v1.5 Signature Verification

1. *Octet string to integer conversion:*
   (a) Reject if bitlength of $S \not\equiv 0 \bmod 8$
   (b) Convert S to integer s
   (c) Reject if $s > n$
2. *RSA computation:* $m = s^e \bmod n$
3. *Integer to octet string conversion:* integer $m \Longrightarrow$ octet string EB
4. *Parsing:* Parse EB into block type BT, padding string PS, and the data D
   (a) Reject if EB cannot be parsed unambigously
   (b) Reject if BT is not 01 (or 00)
   (c) Reject if PS consists of < 8 octets or is inconsistent with BT
5. *Data decoding:*
   (a) BER decode $D \Longrightarrow$ `Message Digest` MD + `DigestAlgorithmIdentifier`
   (b) Reject if `DigestAlgorithmIdentifier` is not allowed
6. *Message digesting and comparison:*
   (a) hash message M to get MD'
   (b) Accept the signature S on M if and only if $MD' = MD$

RSA in Practice — PKCS#1 v2.1 Probabilistic Signature Scheme

- EMSA-PSS-Encode: Probabilistic Encoding of a Message
  1. Generate random salt, approx. 160 bit (for SHA-1)
  2. M' = fixed padding1 || h(M) || salt
  3. H=h(M')
  4. data block DB = fixed padding2 || salt
  5. mask value dbMask = mask generation function MGF(H)
  6. maskedDB = mask value dbMask XOR data block DB
  7. Encoded Message EM = maskedDB || H || fixed padding bc
- Encoded Message goes into RSA function $(\mathtt{EM})^d \bmod n$.
- EMSA-PSS-Verification follows reverse steps to recover salt, then forward steps to recompute and compare H, see later.
- RSA-PSS is provably secure under the random oracle model, it is based on ideas by Bellare and Rogaway (1998)

RSA in Practice — PKCS#1 v2.1 Probabilistic Signature Scheme (2)



EMSA-PSS-Encode: Probabilistic Encoding of Message

RSA in Practice — PKCS#1 v2.1 Probabilistic Signature Scheme (3)

EMSA-PSS-Verify (M, EM, emBits) with M message, EM encoded message, emBits maximal bit length of EM

1. if length (M) greater than input size of hash function, then “inconsistent”
2. mHash=Hash(M), octet string of length hLen
3. if emLen (length of EM in octets) < hLen + sLen (salt) + 2, then “inconsistent”
4. if rightmost octet of EM not 0xbc (padding), then “inconsistent”
5. maskedDB (emLen-hLen-1 octets) || H (hLen octets) = EM
6. if leftmost 8emLen-emBits in maskedDB are not all zero, then “inconsistent”
7. Let dbMask = mask generation function MGF(H,emLen-hLen-1)
8. Let DB = maskedDB XOR dbMask
9. Set leftmost 8emLen - emBits bits of the leftmost octet in DB to zero.
10. If padding 0x01 not correct, then “inconsistent”
11. Let salt be the last sLen octets of DB.
12. Let M’ = (0x)00 00 00 00 00 00 00 00 || mHash || salt ;
13. Let H’ = Hash(M’);
14. If H=H’ then “consistent” else “inconsistent”

RSA in Practice — PKCS#1 v1.5 Encryption

modulus n (length of n in byte: k, $k \geq 12$), data D (length of D in byte $\leq k - 11$)

1. *Encryption-block formatting: Block Type* BT=02
   Padding String PS pseudorandom, nonzero, length of PS: $k-3-length(D) \geq 8$
   Data D
   $$EB = 00||BT||PS||00||D$$
2. *Octet string to integer conversion:* EB $\Longrightarrow$ integer m, $0 \leq m < n$
3. *RSA computation:* $c = m^e \bmod n$
4. *Integer to octet string conversion:* integer $c \Longrightarrow$ octet string C

Example: RSA 1024 bit, $k = 128$ byte, maximal length of D: 117 byte

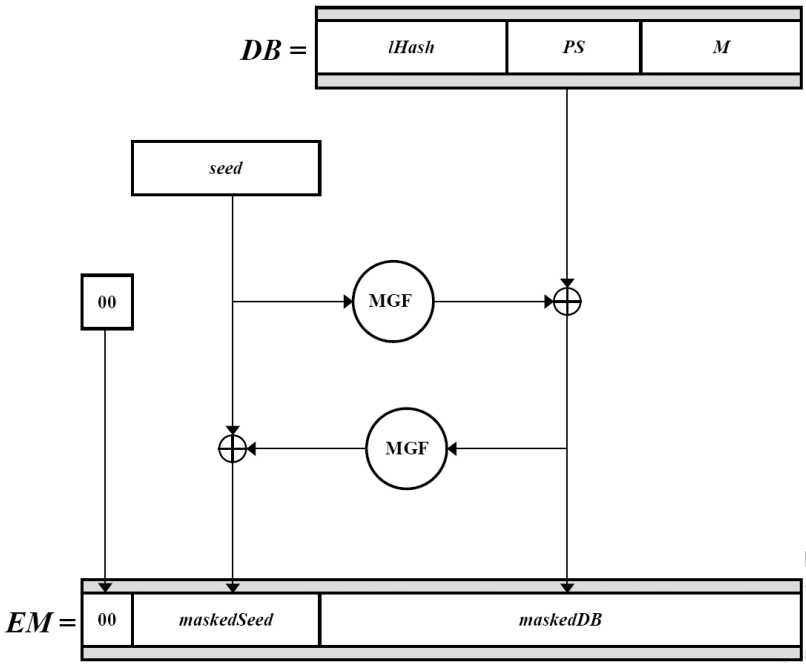
Remark 1.23. Ad hoc countermeasure against Bleichenbacher Attack
In the decryption process, care should be taken to ensure that an attacker cannot distinguish different error conditions while parsing the decoded message.

RSA in Practice — PKCS#1 v2.1 OAEP Encryption

modulus n (length in byte: k), label L, optional, may be empty
hash function with output length $hLen$ octets
message M to be encrypted, $mLen \leq k - 2 \times hLen - 2$
Example: RSA 1024 bit, SHA-1, $k = 128$, $hLen = 20$, $mLen \leq 86$
EME-OAEP Encoding

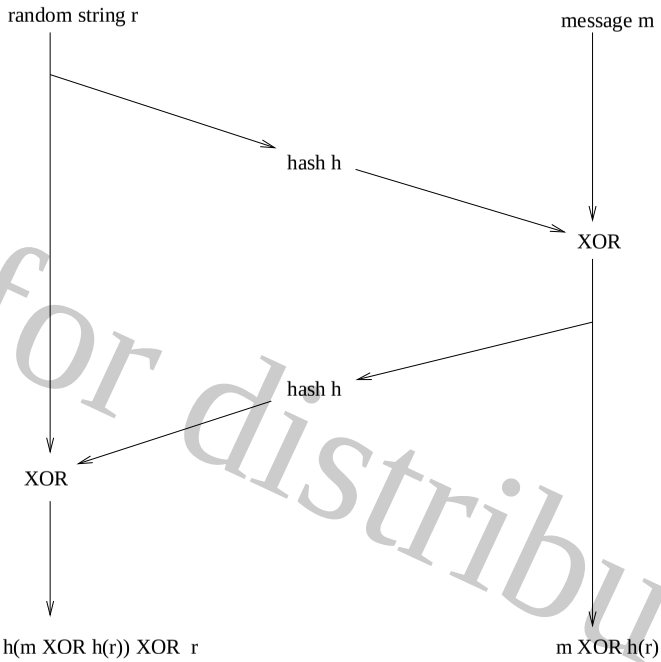
1. Hash(L)=lHash, length $hLen$
2. Padding string of $k - mLen - 2 \times hLen - 2$ zero octets
3. data block $DB = \text{lHash}||PS||01||M$
4. random octet string seed, length $hLen$
5. dbMask = MGF(seed, $k - hLen - 1$)
6. maskedDB = DB XOR dbMask
7. seedMask=MGF(maskedDB, $hLen$)
8. maskedSeed=seed XOR seedMask
9. Encoded Message $EM = 00||\text{maskedSeed}||\text{maskedDB}$ will be used for RSA exponentiation

RSA in Practice — PKCS#1 v2.1 OAEP Encryption (2)



EME-OAEP-Encode

RSA in Practice — PKCS#1 v2.1 OAEP Encryption (3)



Result: h(m XOR h(r)) XOR r || m XOR h(r)

Simplified model EME-OAEP-Encode

RSA in Practice — PKCS#1 v2.1 OAEP Encryption (4)

About the security of OAEP

- OAEP in general was claimed to be IND-CCA2 secure (Bellare, Rogaway 1994)
- But: OAEP in general is only IND-CPA2 secure
- RSA-OAEP is IND-CCA2 secure under the random oracle model (Pointcheval, Stern et al. 2001) but with loose reduction
- OAEP+ (modification by Shoup 2001) is IND-CCA2 with tight reduction
- RSA-OAEP can be proven to be IND-CCA2 in the standard model with stronger assumption
- Hybrid encryption schemes based on the RSA-KEM key encapsulation paradigm offer tight proofs for IND-CCA2

RSA in Practice — Implementation Aspects

What are necessary algorithms for efficient RSA implementations?

- Physical and/or pseudo random number generators for key generation
- Probabilistic prime number tests (Fermat test, Miller-Rabin test)
- Modular multiplication algorithms, Montgomery multiplication or Barrett reduction, Karatsuba or School method
- Garner's algorithm for CRT
- Exponentiation algorithms: Square and Multiply, Window methods
- Euclidean and Extended Euclidean algorithm
- Hash functions
- ASN.1 encoding/decoding
- Encryption: `RSAES-PKCS1-v1_5`, `RSAES-OAEP`
- Signatures: `RSASSA-PKCS1-v1_5`, `RSASSA-PSS`

RSA in Practice — Implementation Aspects — Selected References

- D.E. Knuth: *The Art of Computer Programming*, Vol 2., *Seminumerical Algorithms*, 3rd ed., 1998.
- A. Menezes, P. von Oorschot, S. Vanstone: *Handbook of Applied Cryptography*, 1996.
- Ç.K. Koç: *High-Speed RSA Implementations*. RSA Laboratories, 1994. (nice overview software implementations)
- M. Welschenbach: *Cryptography in C and C++*, 2nd ed., 2005. (introduction!!)
- Tom St. Denis: *BigNum Math. Implementing cryptographic multiple precision arithmetic*, 2006. (author of LibTomCrypt, for programmers)
- *The GNU Multiple Precision Arithmetic Library*, Manual for Edition 4.3.1, 2009.
- E. Hess, B. Meyer, N. Janssen: *Design of Long Integer Arithmetic Units for Public-Key Algorithms*, Proceedings of EUROSMART Security Conference, 2000. (overview hardware designs)
- Ç.K. Koç: *RSA hardware implementation*, RSA Laboratories, August 1995, Version 1.0. (nice overview hardware implementations)

Outline — Section 2: ElGamal / Discrete Logarithms / DSA

1. Diffie-Hellman Key Exchange
2. ElGamal encryption / Discrete Logarithms
3. Attacks on Discrete Logarithm Problems
4. ElGamal signature scheme
5. Digital Signature Algorithm (DSA)

A Note on Notation

The notational conventions used here differ from Stinson's, since many students find Roman letters easier to work with than Greek letters. The following primer showing the correspondence may be useful:

Here	Stinson
p	p
g	α
y	β
x	a
m	x
k	k
r	γ
s	δ

Also, recall that generator is another term for a primitive element.

1. Diffie-Hellman Key Exchange (1)

Remark 2.1.

- Not a public-key cryptosystem
- But basis for ElGamal system
- Diffie-Hellman-(Merkle) protocol already in groundbreaking paper New Directions in Cryptography

Setting:

- Alice and Bob wish to use a symmetric encryption system.
- Initially, they must exchange a secret key.
- DH key exchange system enables them to use the insecure channel for the key exchange.

Diffie-Hellman Key Exchange (2)

- **One-time setup:** An appropriate prime p and generator g of $\mathbb{Z}_p^*$ ($2 \leq g \leq p-2$) are selected and published.
- **Protocol messages:**

  1. $A \rightarrow B$: $g^x \bmod p$ (1)
  2. $B \rightarrow A$: $g^y \bmod p$ (2)

- **Protocol actions:** Perform the following steps each time a shared key is required.

  1. A chooses a random secret x, $1 \leq x \leq p-2$ and sends B message (1).
  2. B chooses a random secret y, $1 \leq y \leq p-2$ and sends A message (2).
  3. B receives g^x and computes the shared key as $K = (g^x)^y \bmod p$.
  4. A receives g^y and computes the shared key as $K = (g^y)^x \bmod p$.

- Unauthenticated DH by itself is vulnerable to Man-in-the-Middle attacks.

2. ElGamal Encryption / Discrete Logarithms

Remark 2.2. History of ElGamal, basic principle

- ElGamal encryption and signature scheme proposed 1984 by Taher ElGamal
- ElGamal signature scheme is the basis of Digital Signature Algorithm (DSA) and its ECC variants EC(G,K)DSA
- ElGamal is probabilistic, thus ciphertext length = two times plaintext length
- Main idea for encryption: use DH to produce message key, mask the message with that key; receiver can complete DH and unmask the message
- ElGamal security depends on the underlying cyclic group
- Other groups than $(\mathbb{Z}_p^*, \times)$ can be used.

ElGamal — Key Generation

Each entity A should do the following:

1. Generate a large random prime p and a generator g of the multiplicative group $\mathbb{Z}_p^*$ of the integers modulo p.
2. Select a random integer x, $1 \leq x \leq p-2$, and compute $g^x \bmod p$.
3. A's public key is $(p, g, g^x \bmod p)$, A's private key is x.

ElGamal — Encryption/Decryption

Encryption: B who encrypts a message m for entity A should do the following:

1. Obtain A's authentic public key $(p, g, g^x \bmod p)$.
2. For encrypting the message m with $0 \leq m < p$ do the following:.
   (a) Select a random integer k, $1 \leq k \leq p-2$.
   (b) Compute $r = g^k \bmod p$ and $s = m \cdot (g^x)^k \bmod p$.
3. Send the ciphertext $c = (r, s)$ to A.

Decryption: To recover the plaintext m from the received ciphertext c, A should do the following:

1. Use the private key x to compute $r^{p-1-x} \bmod p = r^{-x} \bmod p$.
2. Recover m by computing $r^{-x} \cdot s \bmod p$.
   $(r^{-x} \cdot s \equiv g^{-xk} m g^{xk} \equiv m \bmod p.)$

Message mask: $(g^x)^k \bmod p$; Unmasking: Multiply by $g^{-xk} = r^{-x} \bmod p$

Discrete Logarithms

Definition 2.3. Discrete Logarithm Problem (DLP)

The *discrete logarithm problem (DLP)* is the following: given a prime p, a generator g of $\mathbb{Z}_p^*$, and an element $y \in \mathbb{Z}_p^*$, find the unique integer $0 \leq x \leq p-2$, such that $g^x \equiv y \bmod p$.

Remark 2.4.

- The discrete logarithm problem can be defined in any subgroup $\langle g \rangle$ of a multiplicative group $(G, \times)$: Given a multiplicative group $(G, \times)$, an element $g \in G$ having order n and an element $y \in \langle g \rangle$, find the unique integer $0 \leq x \leq n-1$, such that $g^x \equiv y$.
- Usually the finite group G is written in multiplicative form for ElGamal and DSA, but in additive form for elliptic curves:

  Given points $P, Q \in E(F_p)$ with $P = dQ$, find the factor d.

Diffie-Hellman Problems (1)

Definition 2.5. Computational Diffie-Hellman Problem

Consider a multiplicative group $(G, \times)$, an element $g \in G$ having order n and two elements $x, y \in \langle g \rangle$. The problem to find $z \in \langle g \rangle$ such that

$$\log_g z \equiv \log_g x \times \log_g y \bmod n$$

(equivalently, given g^a and g^b , find g^{ab}) is called *computational Diffie-Hellman problem*.

Definition 2.6. Decisional Diffie-Hellman Problem

Consider a multiplicative group $(G, \times)$, an element $g \in G$ having order n and three elements $x, y, z \in \langle g \rangle$. The problem to decide if

$$\log_g z \equiv \log_g x \times \log_g y \bmod n$$

(equivalently, given g^a, g^b, g^c decide if $c \equiv bc \bmod n$) is called *decisional Diffie-Hellman problem*.

Diffie-Hellman Problems (2)

Remark 2.7.

- If the discrete logarithm problem in G can be solved efficiently, then the CDH problem in G is easy.
- If the computational Diffie-Hellman problem CDH in G can be solved efficiently, then the DDH problem in G is easy.
- There are groups in which the discrete log problem and CDH are believed to be hard though the DDH problem is easy.

Diffie-Hellman Problems (3)

Example 2.8. Hardness of Diffie-Hellman problems in certain groups

- It is conjectured that the discrete logarithm problem in the group $\mathbb{Z}_p^*$ is hard.
- The decisional Diffie-Hellman problem in $\mathbb{Z}_p^*$ is not hard (there exist constructive counterexamples).
- But: If p is a strong prime, i. e., $p = 2q + 1$ with q prime, then the cyclic subgroup of $\mathbb{Z}_p^*$ with quadratic residues modulo p has q elements and the DDH problem is believed to be hard in this subgroup.

$$y \in \mathbb{Z}_p^* \text{ quadratic residue mod } p\text{: } \exists x \in \mathbb{Z}_p^* \text{ with } x^2 = y \bmod p.$$

- In modern (post-1980s) cryptography, precise assumptions on which the security relies, are essential.
  $\Longrightarrow$ Don't use an algorithm which is based on DDH assumption in $\mathbb{Z}_p^*$.
  $\Longrightarrow$ ElGamal in the subgroup of quadratic residues mod p of $\mathbb{Z}_p^*$ is conjectured to be semantically secure if discrete logarithm problem in $\mathbb{Z}_p^*$ is infeasible.

ElGamal — Security (1)

Theorem 2.9.

The problem of breaking the ElGamal encryption scheme in $\mathbb{Z}_p^*$, i.e., recovering m given p, g, g^x, r, s is equivalent to solving the computational Diffie-Hellman problem, i.e., constructing the key $K = g^{ab}$ from p, g, g^a, g^b.

Proof 2.10.

(Sketch)

- Suppose Oscar can break the DH key exchange system.
  1. Thus, he can compute the DH key g^{xk} from p, g, g^x, g^k.
  2. Thus, he can reconstruct the message m as $g^{-xk}s \bmod p$.
- Assume Oscar can break ElGamal.
  1. He then applies the decryption algorithm with input $p, g, g^x, r = g^k, s = 1$ and obtains a plaintext m.
  2. He knows that $1 = g^{xk}m \bmod p$.
  3. He can determine the DH key as $g^{xk} \equiv m^{-1} \bmod p$.

ElGamal Security (2)

Remark 2.11.

It is critical, that different integers k be used to encrypt different messages. Otherwise

$$s_1/s_2 = m_1/m_2$$

if m_1 and m_2 are encrypted with same k to $(r, s_1), (r, s_2)$.

Discrete Logarithms — Enumeration

- For $x = 0, 1, 2, \ldots$ compute $g^x \bmod p$ until y is obtained.
- Requires $x - 1$ multiplications and x comparisons in $\mathbb{Z}_p^*$.
- Only the three elements g, y and g^x need to be stored.
- In cryptographic applications we have $x \geq 2^{160}$.

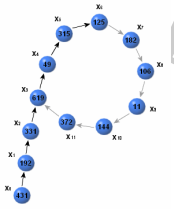
Discrete Logarithms — Shanks' Baby-Step Giant-Step

- Time-memory trade-off of the enumeration method.
- Set $m = \lceil \sqrt{n} \rceil$, where n is the group order and write $x = im + j$, where $0 \leq j < m$. i, j are computed by the algorithm:
  1. We have $g^{im+j} = g^x = y$, i.e., $(g^{-m})^i y = g^j$.
  2. Compute set of baby-steps $B = \{(j, g^j) : 0 \leq j < m\}$.
  3. If in this set, there is a pair (j, y), then $y = g^j$, i.e., $x = j$.
  4. Otherwise determine g^{-m}. For $i = 1, 2, 3, \ldots$ test whether $y(g^{-m})^i$ is the second component of some entry in the table. (giant-step)
  5. If so, we have $y(g^{-m})^i = g^j$, thus $x = im + j$.

  The elements $(g^{-m})^i$ are called giant-steps.
- Algorithm requires storage of $\mathcal{O}(\sqrt{n})$ group elements. Constructing the table takes $\mathcal{O}(\sqrt{n})$ multiplications and $\mathcal{O}(\sqrt{n} \log n)$ operations to sort. The running time of the algorithm is $\mathcal{O}(\sqrt{n})$ group multiplications.

Discrete Algorithms — Pollard rho method

- Pollard rho method, developed in 1978 by John Pollard, works for integer factorization and for discrete logarithms.
- Same complexity $\mathcal{O}(\sqrt{n})$ as Shanks' Baby-Step Giant-Step, but low constant number of storage
- Randomized algorithm, finds cycle in sequence $x_i = g^{a_i} y^{b_i}$
- The sequence x_i enters a loop. Thus, there exists a match with $x_i = x_{i+k}$, k period length. The elements $x_0, \ldots, x_{k-1}$ constitute a pre-cycle.
- This can be visualized in the greek character ρ.



Pollard's-ρ-Algorithm, Source: Wikipedia

Discrete Algorithms — Further Attacks

Remark 2.12.

- Enumeration, Baby-Step Giant-Step, and Pollard's rho method work in arbitrary groups.
- Pohlig-Hellman's algorithm works in arbitrary groups, but is especially efficient if the group order has only small prime factors.
- The index calculus method or number field sieve, a generalization of the quadratic sieve, is efficient only in certain groups. It has subexponential running time, e. g., $L_p[1/3, 1.92]$ for $\mathbb{Z}_p^*$.
- Index calculus methods require a factor base of small primes. This does not exist for elliptic curves!

4. ElGamal Signatures

- Same basic set-up as ElGamal encryption:
- Key generation: generate
  - a prime p such that the discrete logarithm problem in $\mathbb{Z}_p^*$ is (believed) intractible
  - a generator $g \in \mathbb{Z}_p^*$
  - a random integer $1 \leq x \leq p-2$
  - $y = g^x \bmod p$
- Public (verification) key is (p, g, y)
- Private (signing) key is (p, g, x). (We often just call x the private key, since p and g are known from the public key.)

ElGamal Signatures, cont'd

Suppose (p, g, x) is Alice's private key and (p, g, y) is Alice's public key.

- To sign a message $m \in \mathbb{Z}_p^*$, Alice:
  - selects random $k \in \mathbb{Z}_{p-1}^*$, i.e., $k \in \{1, \ldots, p-2\}$ with $\gcd(k, p-1) = 1$.
  - sets $r = g^k \bmod p$ and $s = k^{-1}(m - rx) \bmod (p-1)$.
  - sends (m, r, s) as the signed message.
- To verify a claimed signature (m, r, s), Bob:
  - verifies whether $1 \leq r \leq p-1$. If not, rejects.
  - computes $v = g^m \bmod p$ and $w = y^r r^s \bmod p$
  - accepts if $v = w$. Otherwise rejects.

Remark 2.13.

The ElGamal keys are the same as in ElGamal encryption, but the algorithm is not simply reverse encryption.

Correctness

Proposition: If Alice signed (m, r, s), then $v = w$, i.e., the signature will verify.

Proof: Given a properly signed (m, r, s), we have:

$$\begin{aligned} w &\equiv y^r r^s \pmod{p} \\ &\equiv (g^x)^r (g^k)^s \pmod{p} \\ &\equiv g^{rx} g^{(kk^{-1}(m-rx)) \bmod (p-1)} \pmod{p} \\ &\equiv g^{rx} g^{kk^{-1}(m-rx)} \pmod{p} \text{ since } g^{(p-1)} \equiv 1 \pmod{p} \\ &\equiv g^m \pmod{p} \\ &\equiv v \pmod{p} \qquad \square \end{aligned}$$

Security of ElGamal Signatures: Selective Forgeries

To forge a signature on a given message m without knowing the private key x, an attacker must find r and s such that m, r, and s satisfy the verification equations.

- If the attacker chooses an r and then tries to find the appropriate s, then he must solve the verification equation for s:

$$\begin{aligned} g^m &\equiv y^r r^s \pmod{p} \\ r^s &\equiv g^m y^{-r} \pmod{p} \\ s &\equiv \log_r g^m y^{-r} \pmod{p} \end{aligned}$$

  That is, he must solve a discrete log problem in $\mathbb{Z}_p^*$, which was assumed intractible.
- If the attacker chooses s and then tries to find r, he must solve:

$$y^r r^s \equiv g^m \pmod{p}$$

  for r. No feasible solution is known, but neither is it known equivalent to computing discrete logs.

Security of ElGamal Signatures: Existential Forgeries

- An attacker might somehow be able to somehow compute an r and an s "simultaneously". No such attacks are known, nor a relation to the discrete log problem.
- If he chooses $r = g^k \bmod p$ and s first and tries to solve for m, then he needs m such that:

$$g^m \equiv y^r r^s \pmod{p}, \text{ or}$$
$$m \equiv \log_g y^r r^s \pmod{p},$$

  again a discrete log instance.
- But, there is a key-only existential forgery attack in which m, r, and s are chosen "simultaneously". (See next slide).
- As with RSA, hash functions help to augment resistance to selective forgeries into resistance to existential forgeries.

ElGamal Signatures: Key-Only Existential Forgery

Let i and j be integers between 1 and $p-2$ such that $\gcd(j, p-1) = 1$, and set $r = g^i y^j \bmod p$. The verification condition is:

$$g^m \equiv y^r (g^i y^j)^s \pmod{p}, \text{ or equivalently, } g^{(m-is)} \equiv y^{(r+js)} \pmod{p}.$$

Last equation can be satisfied by, in particular, causing both sides to be equivalent to $1 \bmod p$. By Fermat's Theorem, this will happen if:

$$m - is \equiv 0 \pmod{p-1} \text{ and } r + js \equiv 0 \pmod{p-1} \quad (1)$$

Since $\gcd(j, p-1) = 1$, we can compute $j^{-1} \bmod (p-1)$. Hence (1) can be solved as:

$$m = is \bmod (p-1)$$
$$js = -r \bmod (p-1)$$
$$s = -rj^{-1} \bmod (p-1)$$

then (m, r, s) will pass the verification equations and be considered a valid signature.

ElGamal Signatures: Known Message Existential Forgery

There is also another type of forgery, in which an attacker can create new signatures (on new messages) from old ones.

Suppose an attacker knows a valid signature (m, r, s), and let h, i, and j be integers between 0 and $p-2$ such that $\gcd(hr - js, p-1) = 1$. Set

$$\begin{aligned} \ell &= r^h g^i y^j \bmod p \\ q &= s\ell(hr+js)^{-1} \bmod (p-1) \\ m' &= \ell(hm+is)(hr+js)^{-1} \bmod (p-1) \end{aligned}$$

Then $y^\ell \ell^q \equiv g^{m'} \pmod{p}$, so (m', ℓ, q) is a valid signature.

Remark 2.14.

Adding a hash function prevents these existential forgery attacks from succeeding unless the hash function can be broken. That is, signing $h(m)$ instead of m seems to result in a secure version of ElGamal.

ElGamal Protocol Failures

- Clearly, if the random value k used in computing a signature (m, r, s) is revealed, then it is easy to perform a total break (recover the private key):

$$x = (m - ks)r^{-1} \pmod{(p-1)}$$

- Also, the same random value k should not be reused, as given two signatures (m_1, r, s_1) and (m_2, r, s_2) it is possible to compute k, then yielding the same total break as above:

$$(s_1 - s_2) \equiv k^{-1}(m_1 - m_2) \pmod{(p-1)}$$

- The check $1 \leq r \leq p-1$ in signature verification is essential to avoid existential forgery.
- ElGamal is seldom used directly since the size of signatures (r, s) is two times the bit length of p.

5. Digital Signature Algorithm (DSA)

- **Idea:** Modification of ElGamal that works in a much smaller size q subgroup of $\mathbb{Z}_p^*$ so that signatures are two elements representable by two bitstrings of length $\log q$ (instead of two bitstrings of length $\log p$ like ElGamal), while still retaining security based on solving discrete logarithm problems in $\mathbb{Z}_p^*$. Typically p would be 1024 or more bits, while q would be 160.
- DSA was adopted as a NIST standard in 1994.
- It also incorporates a hash function directly into the encryption rather than requiring the hash-then-sign paradigm. This also lets it sign messages of arbitrary length. (This is true with the hash-then-sign paradigm too.)
- DSA uses a trick by Schnorr working with a qth root of unity to reduce signature sizes.
- There is also an elliptic curve version of DSA, which allows smaller p to be used for equivalent security (given the current state of our knowledge for computing discrete logs in the corresponding groups), improving computation times for computing and verifying signatures.

5. Digital Signature Algorithm (DSA) (2)

- Note (as we will need this for key generation) that it is easy to construct a qth root g of 1 modulo p: let g_0 be a generator of $\mathbb{Z}_p$ and define $g = g_0^{(p-1)/q} \bmod p$. A qth root of 1 is also called a qth root of unity.
- Assume $h : \{0,1\}^* \to \mathbb{Z}_q$ is a secure hash function.
  Key Generation:
  - Let p and q be primes such that $p - 1 \equiv 0 \bmod q$ and the discrete logarithm problem in $\mathbb{Z}_p^*$ is intractible.
  - Let g be a qth root of unity modulo p. (That is, so that $g^q \equiv 1 \bmod p$.)
  - Choose a random $x \in \mathbb{Z}_q$, and compute $y = g^x \bmod p$.
  - public key is (p, q, g, y)
  - private key is x or (p, q, g, x)

Remark 2.15.

Key is like an ElGamal key, except that a qth root of unity is used instead of a generator, and that the private key is chosen in $\mathbb{Z}_q$ instead of $\mathbb{Z}_p$.

Digital Signature Algorithm (DSA) (3)

- For the past standard FIPS 186-2, January 2000,
  - SHA-1 is used as the hash function h
  - q is a 160-bit prime
  - p is an L-bit prime where $L \equiv 0 \pmod{64}$ and $512 \leq L \leq 1024$. In October 2001, NIST recommended only using $L = 1024$.
- For the current standard FIPS 186-3, June 2009,
  - q is an N-bit prime, p is an L-bit prime, $q|(p-1)$
  - Required bit length for p and q and corresponding hash function

L	N	Hash
1024	160	SHA-1
2048	224	SHA-224
2048	256	SHA-256
3072	256	SHA-256

Digital Signature Algorithm (DSA) (4)

Suppose (p, q, g, x) is Alice's private key and (p, q, g, y) is Alice's public key, which is known to Bob with $q|p-1$ and $g \in \mathbb{Z}_p^*$.

- To sign a message m of arbitrary length, Alice:
  - selects a random $k \in \mathbb{Z}_q^*$.
  - sets $r = (g^k \bmod p) \bmod q$.
  - compute $k^{-1} \bmod q$.
  - compute $s = k^{-1}(h(m) + xr) \bmod q$.
  - sends (m, r, s) as the signed message.
- To verify a claimed signature (m, r, s), Bob:
  - verifies whether $1 \leq r, s \leq q-1$. If not, rejects.
  - computes $w = s^{-1} \bmod q$ and $h(m)$.
  - computes $u_1 = wh(m) \bmod q$ and $u_2 = rw \bmod q$.
  - computes $v = (g^{u_1} y^{u_2} \bmod p) \bmod q$.
  - accepts if $v = w$. Otherwise rejects.

Digital Signature Algorithm (DSA) (5)

Remark 2.16.

- The security relies on two discrete logarithm problems: the one in $\mathbb{Z}_p^*$ where the index-calculus is the most powerful method and the one in the cyclic subgroup of order q.
- Signature verification requires two exponentiations modulo p or one multi-exponentiation in contrast to one exponentiation for signature generation, compare with RSA verification which is much cheaper (using $e = 3, F_4$) than RSA signature generation.
- The random ephemeral key k must have very good statistical properties. Bleichenbacher showed 2001 (unpublished) that the Pseudo-RNG proposed in FIPS 182-2 leads to a slight skewness in the bits of k. This can be exploited for an attack.
  $\Longrightarrow$ change notice to FIPS 186-2 in October 2001

Overview: Attack Algorithms on Modern Cryptosystems

Crypto Algorithm	Attack Algorithm	Complexity	Secure Parameter Size
2DES, 3DES	Brute Force	$2^{112}, 2^{168}$	$112^*, 168$
AES with key length k	Brute Force	2^k	$128, 192, 256$
hash function, output length n	Birthday Paradox	$\mathcal{O}(n/2)$	$160^*, 224, 256, 384, 512$
RSA, modulus N, length n	Number Field Sieve	$L_N[\frac{1}{3}, 1.923]$	$1024^*, 2048, 4096$
DSA with p (field) and q (order of subgroup)	Index Calculus or Number Field Sieve in F_p	$L_p[\frac{1}{3}, 1.923]$	$1024^*, 2048, 4096$
	Pollard rho in subgroup	$\mathcal{O}(\sqrt{q})$	$160^*, 224, 256, 384, 512$
ECC over F_p, group order n	Pollard rho	$\mathcal{O}(\sqrt{n})$	$160^*, 224, 256, 384, 521$

Attack Complexities and Secure Parameter Sizes for Modern Cryptosystems, * should not be used for new systems, no long time security