Radbound University

RSA

Cryptography, Spring 2020

L. Batina, J. Daemen

May 27, 2020

Institute for Computing and Information Sciences
Radboud University

Outline

Euler totient function

$((\mathbb{Z}/n\mathbb{Z})^*, \times)$ with n not prime

Remember

Invertibility criterion

m has multiplicative inverse modulo n (i.e., in $\mathbb{Z}/n\mathbb{Z}$) iff $\gcd(m, n) = 1$

- We define $(\mathbb{Z}/n\mathbb{Z})^* = \{m \mid m \in \mathbb{Z}/n\mathbb{Z} \text{ and } \gcd(m, n) = 1\}$
- $((\mathbb{Z}/n\mathbb{Z})^*, \times)$ is an abelian group
  - closed: if $\gcd(a, n) = 1$ and $\gcd(b, n) = 1$, then $\gcd(ab, n) = 1$
  - 1 is neutral element
  - each element in $(\mathbb{Z}/n\mathbb{Z})^*$ has an inverse
  - associativity and commutativity follow from multiplication in $\mathbb{Z}$
- But what is the order of $(\mathbb{Z}/n\mathbb{Z})^*$? (we will need that!)

  This is *Euler's totient function*

Computing the order of $(\mathbb{Z}/n\mathbb{Z})^*$

Definition: Euler's totient function

Euler's totient function of an integer n, denoted $\varphi(n)$, is the number of integers smaller than and coprime to n

- For prime p, all integers 1 to $p-1$ are coprime to p: $\varphi(p) = p - 1$
- If $n = a \cdot b$ with a and b coprime: $\varphi(a \cdot b) = \varphi(a)\varphi(b)$
- For the power of a prime p^n: $\varphi(p^n) = (p-1)p^{n-1}$
- Computing $\varphi(n)$:
  - factor n into primes and their powers
  - apply $\varphi(p^n) = (p-1)p^{n-1}$ to each of the factors
- Example: $\varphi(2020) = \varphi(2^2 \cdot 5 \cdot 101) = 2 \cdot 4 \cdot 100 = 800$

Fact: **computing $\varphi(n)$ is as hard as factoring n (see lecture notes)**

Euler's theorem (Leonhard Euler, 1736)

If $\gcd(x, n) = 1$, then $x^{\varphi(n)} \equiv 1 \bmod n$

PROOF:

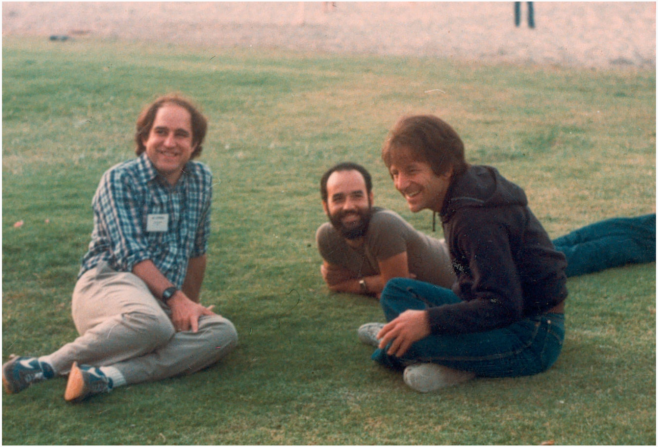
If $\gcd(x, n) = 1$, then $x \in (\mathbb{Z}/n\mathbb{Z})^*$
We know $\#(\mathbb{Z}/n\mathbb{Z})^* = \varphi(n)$
Lagrange says: $\mathrm{ord}(x)$ divides $\varphi(n)$
Therefore $x^{\varphi(n)} \bmod n = 1$ $\square$

We can use this for computing inverses in $(\mathbb{Z}/n\mathbb{Z})^*$ with exponentiation:

$$x^{-1} = x^{\varphi(n)-1} \bmod n$$

…just as we did in $(\mathbb{Z}/p\mathbb{Z}) \setminus \{0\}$

The RSA cryptosystem



Designed their famous cryptosystem in 1977

What is the RSA cryptosystem?

RSA is a *trapdoor one-way function* $y = f(x)$

(1) given x, computing $y = f(x)$ is easy
(2) given y, finding x is difficult
(3) given y and trapdoor info: computing $x = f^{-1}(y)$ is easy

Concretely:

(1) $f(x) = x^e \bmod n$ with $n = pq$ with p, q primes and $\gcd(e, \varphi(n)) = 1$
(2) $f^{-1}(y) = y^d \bmod n$ requires knowing d with $ed \equiv 1 \bmod \varphi(n)$
(3) Trapdoor info: d, or equivalently $\varphi(n)$, or equivalently p and q

Public and private keys:

(1) Public key: (n, e)
(2) n is the product of two large primes and different for each public key
(3) Private key: (n, d)

Domain parameters? RSA has none! (except maybe the fixed value of e)

How does RSA work?

- Why is $x = y^d$ when $y = x^e$? (we omit $\bmod n$ for brevity)
  (1) Substitution gives $y^d = (x^e)^d = x^{ed}$
  (2) Euler's theorem says $x^{\varphi(n)} = 1$ so $x^{ed} = x^{ed \bmod \varphi(n)}$
  (3) By the definition of d we have $ed \bmod \varphi(n) = 1$
  (4) It follows $x^{ed \bmod \varphi(n)} = x$
- Computation of d from e and p, q
  - inverse of e modulo $\varphi(n) = \varphi(pq) = (p-1)(q-1)$
  - it only exists if $\gcd(e, p-1) = 1$ and $\gcd(e, q-1) = 1$
  - just apply extended Euclidean algorithm to $(p-1)(q-1)$ and e

Quiz questions:

(1) *can we compute d by exponentiation?*

(2) if so, what would be the base, exponent and modulus?

Textbook RSA encryption and signing

- Encryption of a message $m \in (\mathbb{Z}/n\mathbb{Z})^*$
  - Bob uses (n, e) to encipher m to cryptogram $c = m^e$ for Alice
  - Alice deciphers cryptogram c with (n, d) to $m = c^d$

Security breaks down if Eve can find the e^{th} root of c

- Signing a message $m \in (\mathbb{Z}/n\mathbb{Z})^*$
  - Alice signs message m with (n, d): signature $s = m^d$ over
  - Bob (or anyone) verifies s using (n, e) as $m \stackrel{?}{=} s^e$

Security breaks down if Eve can find an e^{th} root of some chosen m

- Knowing $\varphi(n)$ allows computing d and hence finding an e^{th} root

$\Rightarrow$ the security of textbook RSA requires factoring to be hard

Converse is not true: textbook RSA is non-secure even if factoring is hard

Chinese remainder theorem

Something uneasy with our usage of RSA

- When encrypting m we must take $m \in (\mathbb{Z}/n\mathbb{Z})^*$
  - but we don't know $(\mathbb{Z}/n\mathbb{Z})^*$
  - that would require knowing p and q and hence the private key
  - best we can do is choose $m \in (\mathbb{Z}/n\mathbb{Z}) \setminus \{0\}$
  - this set has $(pq - 1) - (p - 1)(q - 1) = p + q$ elements that are not in the group
- What happens when we compute $c \leftarrow m^e$ with m one of these?
  - choosing such an m only happens with probability $(p + q)/pq$
  - still interesting to know: what if?
- It turns out to be no problem: c^d will yield the original m
  - are we lucky or is this coincidence?
  - the world of algebra knows no luck or coincidence
- It can be explained with the help of the Chinese Remainder Theorem

Cross-product of rings

Definition of cross product of groups

Given groups $(G, *)$ and $(H, \circ)$, the cross product group $(G \times H, \cdot)$ has

set: $\{(g, h) \mid g \in G, h \in H\}$

group operation: $(g, h) \cdot (g', h') = (g * g', h \circ h')$

The same can be applied to cross-product of rings, in particular

Cross-product of rings of integers modulo n

Given $(\mathbb{Z}/n_1\mathbb{Z}, +, \times)$ and $(\mathbb{Z}/n_2\mathbb{Z}, +, \times)$, the cross product ring $(\mathbb{Z}/n_1\mathbb{Z} \times \mathbb{Z}/n_2\mathbb{Z}, +, \times)$ has

set: $\{(g, h) \mid g \in \mathbb{Z}/n_1\mathbb{Z}, h \in \mathbb{Z}/n_2\mathbb{Z}\}$

addition: $(g, h) + (g', h') = (g + g' \bmod n_1, h + h' \bmod n_2)$

multiplication: $(g, h) \times (g', h') = (g \times g' \bmod n_1, h \times h' \bmod n_2)$

This generalizes to the cross-product of more than two groups or rings

Chinese Remainder Theorem (general)

Chinese Remainder Theorem (CRT)

Let $n = n_1 \cdot n_2 \cdots n_k$ with all n_i, n_j coprime, then the map

$$x \mapsto (x_1, x_2, \ldots, x_k) \text{ with } x \in \mathbb{Z}/n\mathbb{Z} \text{ and } \forall i : x_i = x \bmod n_i$$

defines a ring isomorphism:

$$\mathbb{Z}/n\mathbb{Z} \cong \mathbb{Z}/n_1\mathbb{Z} \times \mathbb{Z}/n_1\mathbb{Z} \times \ldots \times \mathbb{Z}/n_k\mathbb{Z}$$

Informally, any sum or product of elements in $\mathbb{Z}/n\mathbb{Z}$ is matched by that of the corresponding elements in $\mathbb{Z}/n_1\mathbb{Z} \times \mathbb{Z}/n_2\mathbb{Z} \times \ldots \times \mathbb{Z}/n_k\mathbb{Z}$

Usually the term CRT is used for computing x from $(x_1, x_2, \ldots, x_k)$

Chinese Remainder Theorem (specific for RSA)

Chinese Remainder Theorem (CRT)

Let $n = p \cdot q$ with p, q primes, then the map

$$x \mapsto (x_1, x_2) \text{ with } x \in \mathbb{Z}/n\mathbb{Z}, x_1 = x \bmod p \text{ and } x_2 = x \bmod q$$

defines a ring isomorphism:

$$\mathbb{Z}/n\mathbb{Z} \cong \mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z}$$

Informally, any sum or product of elements in $\mathbb{Z}/n\mathbb{Z}$ is matched by that of the corresponding elements in $\mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z}$

Usually the term CRT is used for computing x from (x_1, x_2)

CRT visually for $n = 77, p = 11, q = 7$

	0	1	2	3	4	5	6	7	8	9	10
0	0			14				7			21
1	22	1			15				8		
2			2			16				9	
3				3			17				10
4	11				4			18			
5		12				5			19		
6			13				6			20	

CRT visually for $n = 77, p = 11, q = 7$, complete

	0	1	2	3	4	5	6	7	8	9	10
0	0	56	35	14	70	49	28	7	63	42	21
1	22	1	57	36	15	71	50	29	8	64	43
2	44	23	2	58	37	16	72	51	30	9	65
3	66	45	24	3	59	38	17	73	52	31	10
4	11	67	46	25	4	60	39	18	74	53	32
5	33	12	68	47	26	5	61	40	19	75	54
6	55	34	13	69	48	27	6	62	41	20	76

Chinese Remainder Theorem, alternative version (general)

Chinese Remainder Theorem (CRT), alternative version

If $n = \prod_i n_i$ with $n_1, n_2, \dots, n_k$ pairwise coprime integers, then the system of congruence relations:

$$x \equiv x_i \pmod{n_i}, 1 \leq i \leq k,$$

has a unique solution $x \in \mathbb{Z}/n\mathbb{Z}$ for any k-tuple of integers $(x_1, x_2, \dots, x_k)$

The mapping from x to $(x_1, x_2, \dots, x_k)$ is injective: different values x cannot give equal tuples $(x_1, x_2, \dots, x_k)$

The number of possible values for x and $(x_1, x_2, \dots, x_k)$ is both n and hence the mapping is a bijection

Chinese Remainder Theorem, alternative version (RSA-specific)

Chinese Remainder Theorem (CRT), alternative version

If $n = p \cdot q$ with p, q primes, then the system of congruence relations:

$$x \equiv x_1 \pmod{p}$$
$$x \equiv x_2 \pmod{q}$$

has a unique solution $x \in \mathbb{Z}/n\mathbb{Z}$ for any couple of integers (x_1, x_2)

The mapping from x to (x_1, x_2) is injective: different values x cannot give equal tuples (x_1, x_2)

The number of possible values for x and (x_1, x_2) is both n and hence the mapping is a bijection

CRT formula (general)

CRT formula

The solution $x \in \mathbb{Z}/n\mathbb{Z}$ with $n = \prod_i n_i$ for

$$x \equiv x_i \pmod{n_i}, 1 \leq i \leq k,$$

with $n_1, n_2, \ldots, n_k$ pairwise coprime integers is given by

$$x = u_1 x_1 + u_2 x_2 + \ldots + u_k x_k \bmod n$$

with $\forall i$: $u_i = r \cdot (n/n_i)$ with $r = (n/n_i)^{-1} \bmod n_i$

It can be seen that for all i, u_i satisfies following equations:

$$u_i \equiv 1 \pmod{n_i} \text{ for all } i$$
$$u_i \equiv 0 \pmod{n_j} \text{ for all } i \neq j$$

The constants u_i can be used for any vector $(x_1, x_2, \ldots, x_k)$

Computing and using the CRT formula: example

Let $n = 616 = 7 \cdot 11 \cdot 8$

Computation of the constants u_i:

n_i	n/n_i	$\bmod n_i$	inverse	u_i	$u_i \bmod 7$	$u_i \bmod 11$	$u_i \bmod 8$
7	88	4	2	176	1	0	0
11	56	1	1	56	0	1	0
8	77	5	5	385	0	0	1

Computing x for some vectors x_i

(x_1, x_2, x_3)	expression	x
$(2, 4, 1)$	$2 \cdot 176 + 4 \cdot 56 + 1 \cdot 385 \bmod 616$	114
$(3, 3, 3)$	$3 \cdot 176 + 3 \cdot 56 + 3 \cdot 385 \bmod 616$	3
$(1, 1, 0)$	$1 \cdot 176 + 1 \cdot 56 + 0 \cdot 385 \bmod 616$	232
$(6, 10, 7)$	$6 \cdot 176 + 10 \cdot 56 + 7 \cdot 385 \bmod 616$	615

CRT formula (RSA-specific)

CRT formula

The solution $x \in \mathbb{Z}/n\mathbb{Z}$ with $n = pq$ for

$$x \equiv x_1 \pmod{p}$$
$$x \equiv x_2 \pmod{q}$$

with p, q primes is given by

$$x = u_1 x_1 + u_2 x_2 \bmod n$$

with $u_1 = \left(q^{-1} \bmod p\right) \cdot q$ and $u_2 = \left(p^{-1} \bmod q\right) \cdot p$

It can be seen that:

$$u_1 \equiv 1 \pmod{p} \qquad u_1 \equiv 0 \pmod{q}$$
$$u_2 \equiv 0 \pmod{p} \qquad u_2 \equiv 1 \pmod{q}$$

The constants u_i can be used for any vector (x_1, x_2)

For the two-factor case the CRT formula can be simplified

Garner's algorithm (Harvey Garner, 1959)

INPUT: (p, q) with $p > q$ and (x_1, x_2),
OUTPUT: x
$i_q = q^{-1} \bmod p$
$t = x_1 - x_2 \bmod p$
$x = x_2 + q \cdot (t \cdot i_q \bmod p)$

Verify that this is correct!

RSA private key exponentiation in the cross-product ring

Given y we must compute x that satisfies $y = x^e \bmod pq$

For $(x_1, x_2) \in \mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z}$ we get $y_1 = x_1^e \bmod p$ and $y_2 = x_2^e \bmod q$

These are solved by

- $x_1 \leftarrow y_1^{d_p} \bmod p$ with d_p the solution of $ed_p \equiv 1 \pmod{p-1}$
- $x_2 \leftarrow y_2^{d_q} \bmod q$ with d_q the solution of $ed_q \equiv 1 \pmod{q-1}$

This works for **all** values of y_1 and y_2 including 0 (**Check this!**)

Thanks to CRT, it follows that $x \leftarrow y^d \bmod n$ always works, with

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

Note that one cannot compute d from d_p and d_q using CRT (Why not?)

RSA CRT private key operation with Garner

RSA with Garner's algorithm

INPUT:

- base c
- private key $p, q, d_p, d_q, i_q (= q^{-1} \bmod p)$

OUTPUT: m

(1) $c_1 \leftarrow c \bmod p,\ m_p \leftarrow {c_1}^{d_p} \bmod p$

(2) $c_2 \leftarrow c \bmod q,\ m_q \leftarrow {c_2}^{d_q} \bmod q$

(3) $t \leftarrow m_p - m_q \pmod{p}$

(4) $m \leftarrow m_q + q \cdot (t \cdot i_q \bmod p)$

Efficiency gain from using CRT

- moving addition from $\mathbb{Z}/n\mathbb{Z}$: $x + y \bmod n$ to $\mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z}$:
  - $x_1 + y_1 \bmod p$
  - $x_2 + y_2 \bmod q$

  similar efficiency: two short additions instead of one long
- moving multiplication from $\mathbb{Z}/n\mathbb{Z}$: $x \cdot y \bmod n$ to $\mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z}$:
  - $x_1 \cdot y_1 \bmod p$
  - $x_2 \cdot y_2 \bmod q$

  factor 2 more efficient: two short multiplications instead of one long
- moving exponentiation from $\mathbb{Z}/n\mathbb{Z}$: $x^d \bmod n$ to $\mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z}$:
  - ${x_1}^d \bmod p$ or ${x_1}^{d \bmod p-1} \bmod p$
  - ${x_2}^d \bmod q$ or ${x_2}^{d \bmod q-1} \bmod q$

  factor 4 more efficient: two short exponentiations instead of one long

So use of CRT speeds up RSA private key exponentiation with a factor 4

On the choice of d

Fact: $\forall x_1 \in \mathbb{Z}/p\mathbb{Z}$, $\mathrm{ord}(x_1) \mid (p-1)$ and $\forall x_2 \in \mathbb{Z}/q\mathbb{Z}$, $\mathrm{ord}(x_2) \mid (q-1)$

So $\forall (x_1, x_2) \in \mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z}$, $\mathrm{ord}((x_1, x_2)) \mid \mathrm{lcm}(p-1)(q-1)$

With *least common multiple* $\mathrm{lcm}(a, b) = a \cdot b / \gcd(a, b)$

Thanks to CRT this also holds for elements of $(\mathbb{Z}/pq\mathbb{Z})^*$

This implies we can compute d as the inverse of e modulo $\mathrm{lcm}(p-1, q-1)$ instead of modulo $(p-1)(q-1)$

This is what standards prescribe (e.g. NIST FIPS 186)

Toy example: $p = 13, q = 17$, $\mathrm{ord}(x_1) \mid 12$, $\mathrm{ord}\, x_2 \mid 16$
so $\mathrm{ord}((x_1, x_2)) \mid 48$ while $(p-1)(q-1) = 192$

RSA key pair generation

RSA key pair generation

Generating an RSA key pair with given modulus length $|n| = \ell$:

- $|n|$ determines security of RSA key pair, but also efficiency
  - No consensus on how to choose length
  - See `www.keylength.com` for advice by *experts*

Procedure to generate an RSA key pair:

(1) choose e: often this is fixed to $2^{16}+1$ by the context

(2) randomly choose prime p with $|p| = \ell/2$ and $\gcd(e, p-1) = 1$

(3) randomly choose prime q such that $|pq| = \ell$ and $\gcd(e, q-1) = 1$

(4) compute modulus $n = p \cdot q$

(5) compute the private key exponent(s)
  - no CRT: $d \leftarrow e^{-1} \bmod \operatorname{lcm}(p-1, q-1)$
  - CRT: $d_p \leftarrow e^{-1} \bmod (p-1)$, $d_q \leftarrow e^{-1} \bmod (q-1)$, $i_q \leftarrow q^{-1} \bmod p$

Generation of a random prime with a given length

Method: randomly generate ℓ-bit integer x then increment until (probably) prime

Input: length ℓ and public exponent e
Output: (probable) prime p
generate $\ell - 2$ random bits, put a 1 before and after
interpret the result as an integer x: odd integer length ℓ
repeat
  if $\gcd(x-1, e) = 1$ **then**
    randomly choose $b \in \mathbb{Z}/x\mathbb{Z}$
    if $(b^{x-1} \bmod x = 1)$ (Fermat: holds if x prime and likely not otherwise) **then**
      do w more Fermat tests for randomly chosen b
      if all tests pass **then**
        return $p = x$
      else
        $x \leftarrow x + 2$
    else
      $x \leftarrow x + 2$
  else
    $x \leftarrow x + 2$
until false

This is an example, there are several other approaches

Distribution of prime numbers

There are infinitely many primes (Euclid, 300 BC)

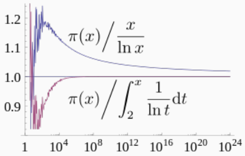
prime counting function $\pi(n)$

$\pi(n) = \#p_i, p_i \leq n$, where p_i is a prime

For example $\pi(100) = 25$

Prime number theorem (mathematicians, XVIII century - today)

$$\lim_{n\to\infty} \frac{\pi(n)}{n/\ln n} = 1 \tag{1}$$



Consequence: expected distance between ℓ-bit primes is close to $\ell \ln 2$

Generation of random primes: attention points

- Execution time: long and variable
  - takes multiple exponentiations
  - number of them depends on the distance from x to next prime p
  - expected value is $(\ell \ln 2)/2$ but varies a lot
- Optimization
  - trial division by small primes: $3, 5, 7, 11, \cdots$
  - fixing the base b to small numbers: $2, 3, \ldots$
  - variant of Fermat test: *Rabin-Miller*, slightly more efficient
- Efficiency of RSA key generation
  - expected cost $\approx$ 30 RSA private key operations
  - in concrete cases it can be 5 but also 120
- Security
  - result may be non-prime but probability decreases with number of Rabin-Miller tests
  - **unpredictability of random generator is crucial!**

Security strength of RSA

RSA security: advances of factoring over time

- State of the art of factoring: two important aspects
  - reduction of computing cost: Moore's Law
  - improvements in factoring algorithms
- Factoring algorithms
  - Sophisticated algorithms involving many subtleties
  - Two phases:
    - distributed phase: equation harvesting
    - centralized phase: equation solving
  - Best known: general number field sieve (GNFS)
- These advances lead to increase of advised RSA modulus lengths
  make sure to check `http://www.keylength.com/`

Factoring records

number	digits	date	sieving time	alg.
C116	116	mid 1990	275 MIPS years	mpqs
RSA-120	120	June, 1993	830 MIPS years	mpqs
RSA-129	129	April, 1994	5000 MIPS years	mpqs
RSA-130	130	April, 1996	1000 MIPS years	gnfs
RSA-140	140	Feb., 1999	2000 MIPS years	gnfs
RSA-155	155	Aug., 1999	8000 MIPS years	gnfs
C158	158	Jan., 2002	3.4 Pentium 1GHz CPU years	gnfs
RSA-160	160	March, 2003	2.7 Pentium 1GHz CPU years	gnfs
RSA-576	174	Dec., 2003	13.2 Pentium 1GHz CPU years	gnfs
C176	176	May, 2005	48.6 Pentium 1GHz CPU years	gnfs
RSA-200	200	May, 2005	121 Pentium 1GHz CPU years	gnfs
RSA-768	232	Dec., 2009	2000 AMD Opteron 2.2 Ghz CPU years	gnfs

RSA-240	795 bits	Dec 2, 2019	900 core-years on 2.1 GHz Intel Xeon Gold 6130
RSA-250	829 bits	Feb 28, 2020	

Using RSA

Using RSA for encryption: attention points

Plaintext m shall have enough entropy:

- Otherwise, Eve can guess m and check if $c = m^e \bmod n$

Example: PIN encryption in EMV (Visa, Mastercard) payment cards

- Requirement: protecting PIN in transfer from terminal to card
- Solution: encryption between terminal and smart card using RSA
- *Enhancements:*
  - terminal adds entropy with random string r: $m \leftarrow PIN; r$
  - for freshness: include challenge c from card $m \leftarrow PIN; r; c$

There are many ways to get RSA encryption wrong

Advice: just don't encrypt data with RSA

Using RSA for encryption: solutions

- Apply a hybrid scheme:
  - use RSA for encrypting a symmetric key K
  - encipher (and authenticate) with symmetric cryptography
- Sending an encrypted key
  - addition of redundancy and randomness before encryption
  - verification of redundancy after decryption
  - if NOK, return error
- Many proposals:
  - best known standard: PKCS #1 v1.5 and v2 (e.g. OAEP)
  - rather complex and no consensus on their security
- Despite the problems, this is still the most widespread method

Using RSA for key exchange: state-of-the-art

RSA Key Encapsulation Method (KEM)

- Bob randomly generates $r \in \mathbb{Z}/n\mathbb{Z}$
- Bob sends $c = r^e \bmod n$ to Alice
- Alice deciphers c back to $r = c^d \bmod n$
- both compute shared symmetric key K as $K = \mathrm{h}(\text{“KDF”}; r)$

RSA-KEM is the sound way to use RSA for exchanging a key

Problems of textbook RSA signatures

- RSA *malleability*
  - given two signatures $s_1 = m_1^d$ and $s_2 = m_2^d$, Eve can construct a signature for $m_3 = m_1 \cdot m_2 \bmod n$ by computing $s_3 = s_1 \cdot s_2 \bmod n$.

$$ {m_3}^d = (m_1 \times m_2)^d = {m_1}^d \times {m_2}^d = s_1 \times s_2 $$

  - this is forgery: signing without knowing private key
- Limitation on message length
- Several other attention points

Using RSA for signatures

- Let $\mathsf{h}()$ be a function with co-domain $\mathbb{Z}/n\mathbb{Z}$
- Alice signs message m with her private key: $s \leftarrow (\mathsf{h}(m))^d \bmod n$
- Bob verifies the signed message (m, s):
  - computes $z \leftarrow s^e \bmod n$
  - checks that $z = \mathsf{h}(m)$
- this is secure if the hash function behaves like a random oracle
- This never made it to the standards
  - $\mathcal{RO}$ assumption conflicts with beliefs of many cryptographers
  - requires long hash output and XOFs are reasonably recent
- Most important standards: PKCS $\#$ 1 v1.5 or v2 (PSS)
  - First hashes message $h = \mathsf{h}(m)$ with classical hash function
  - then embeds h into the RSA input in $\mathbb{Z}/n\mathbb{Z}$ ...
  - ... uses padding and some messy processing
  - uses hash function calls to destroy malleability

RSA vs ECC

Computational efficiency of RSA

- Public exponentiation is light (assuming $e = 2^{16} + 1$))
  - 15 squarings and 1 multiplication of $|n|$-bit integers
  - time grows only quadratically with $|n|$
- Private exponentiation is heavy
  - without CRT: $|n|$ $|n|$-bit squarings and multiplications
  - with CRT: $|n|$ $|n|/2$-bit squarings and multiplications
  - time grows with the third power of $|n|$
- Key generation is a nightmare
  - its computation time is unpredictable and has huge variance
  - expected time: about 30 times that of private exponentiation
  - time grows with more than third power of $|n|$

RSA vs ECC

- Disclaimer: fair comparison is probably not possible
  - worse: almost all comparisons out there have a hidden agenda
  - we try to give here advantages and downsides of both
  - keep these in mind when comparing
- For making things concrete we target 128 bits of security
  - ECC: $|p| = 256$ following general consensus
  - RSA: $|n| = 3072$ following advice on `keylength.com`

key lengths	RSA		ECC	
domain parameters	e:	17	p, a, b, G, q, h:	≈ 1400
public key	n:	3072	A:	512
compressed	-		A:	257
private key	d:	3072	a:	256
with Garner	p, q, d_p, d_q, i_q:	3840	-	
compressed	p:	768	-	

RSA signatures vs EC Schnorr signatures

- Computation
  - ECC faster in generation, RSA faster in verification
  - RSA best choice for
    - long-term certificates as in a PKI
    - broadcast signatures as in software updates
  - ECC best choice for
    - certificates over short-lived keys
    - challenge-response entity authentication
- Signature size: ECC 512 bits, RSA 3072 bits
  - but: RSA support *data recovery*
  - inclusion of part of signed message in the signature
  - specified in ISO 9796-2 and used in EMV card certificates
  - overhead can be reduced to about 256 bits

RSA-KEM vs ECDH

- Computation
  - RSA-KEM: light on sending side and heavy on receiving
  - ECDH has same workload on both sides
  - forward secrecy requires generation of fresh key pairs
  - RSA-KEM best choice if
    - sender is lightweight and receiver is not
    - there is some RSA legacy
  - ECDH best choice if
    - forward secrecy is a requirement
    - sender and receiver have similar CPU power
- Data exchanged:
  - there are many cases
  - RSA-KEM with receiver having authentic public key: 3072 bits
  - unilaterally authenticated forward-secret ECDH: 1300 bits

Conclusions

Conclusions

- Until recently, RSA was the most widespread public key crypto
- It remains an amazing cryptosystem
  - underlying mathematics are very interesting
  - supports key exchange, signatures, and much more
- RSA is considered less *cool* than ECC but has unique advantages
  - faster public key operation
  - shorter signature overhead when using data recovery
- But actually, most applications don't require public key crypto
  - just use symmetric crypto
  - orders of magnitudes faster
  - 128-bit keys and tags