RSA

Cryptography, Autumn 2021

Lecturers: J. Daemen, B. Mennink

December 7, 2021

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 n 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^k: $\varphi(p^k) = (p-1)p^{k-1}$
- Computing $\varphi(n)$:
  - factor n into primes and their powers
  - apply $\varphi(p^k) = (p-1)p^{k-1}$ to each of the factors
- Example: $\varphi(2021) = \varphi(47 \cdot 43) = 46 \cdot 42 = 1932$

Fact: hardness of computing the Euler totient function

Computing $\varphi(n)$ is as hard as factoring n (see lecture notes)

Euler's theorem

Euler's theorem (Leonhard Euler, 1736)

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

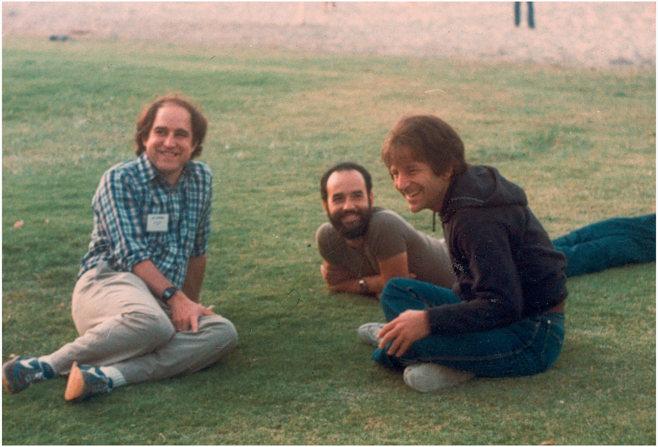
We can use this for computing inverses in $(\mathbb{Z}/n\mathbb{Z})^*$ with exponentiation:

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

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

The RSA cryptosystem

Ron Rivest, Adi Shamir, Leonard Adleman



Designed their famous cryptosystem in 1977

Textbook RSA encryption and signing

Keys: public key (n, e) and private key (n, d) with

- modulus $n = pq$ with p and q two large primes
- public exponent e that satisfies $\gcd(e, \varphi(n)) = 1$
- private exponent d with $ed \equiv 1 \bmod \varphi(n)$

Bob encrypts a message $m \in (\mathbb{Z}/n\mathbb{Z})^*$ for Alice

Bob		Alice
Alice's public key (n, e)		Alice's private key (n, d)
$c \leftarrow m^e \bmod n$	$\xrightarrow{c}$	$m' \leftarrow c^d \bmod n$

Alice signs a message $m \in (\mathbb{Z}/n\mathbb{Z})^*$

Alice		Bob (or anyone)
Alice's private key (n, d)		Alice's public key (n, e)
$s \leftarrow m^d \bmod n$	$\xrightarrow{\text{Alice}, m, s}$	$m \stackrel{?}{=} s^e \bmod n$

Note: RSA has no domain parameters

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 $\varphi(n) = (p-1)(q-1)$
  - it only exists if $\gcd(e, p-1) = 1$ and $\gcd(e, q-1) = 1$
  - just apply extended Euclidean alg. 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?

Security of textbook RSA

Security of textbook RSA:

- Encryption breaks down if Eve can find the e^{th} root of c
- Signing breaks down if Eve can find the e^{th} root of some chosen m
- We call this *inverting RSA*

Security of textbook RSA requires factoring to be hard

- Having the factorization of n allows computing $\varphi(n)$
- Knowing $\varphi(n)$ allows computing d and hence inverting RSA

Converse is not true: textbook RSA is actually 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

Product of rings

Definition of product of groups

Given groups $(G, *)$ and $(H, \circ)$, the 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 product of rings, in particular

Product of rings of integers modulo n

Given $(\mathbb{Z}/n_1\mathbb{Z}, +, \times)$ and $(\mathbb{Z}/n_2\mathbb{Z}, +, \times)$, the 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 product of more than two groups or rings

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 (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 (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 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 $y_2 = x_2^e \bmod q$ (with $y_1 = y \bmod p$ and $y_2 = y \bmod q$)

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 it is not straightforward to 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:

- *ciphertext* 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) $t \leftarrow (m_p - m_q) \bmod p$

(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

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 (or standard)

(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: $d \leftarrow e^{-1} \bmod (p-1)(q-1)$ (or $\mathrm{lcm}(p-1, q-1)$)
  - 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 of given length [for info only]

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)

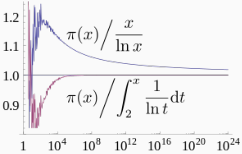
prime counting function $\pi(n)$

$\pi(n) = \#p_i, p_i \leq n$, 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: *Miller-Rabin*, 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 Miller-Rabin 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/`

For 128 bits of security, NIST currently advises 3072-bit modulus

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

Textbook RSA encryption:

Bob has Alice's public key (n, e)		Alice with private key (n, d)
$c \leftarrow m^e \bmod n$	$\xrightarrow{c}$	$m \leftarrow c^d \bmod n$

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, MC) contactless payment

- Requirement: protecting PIN between terminal to card
- Solution: terminal encrypts PIN with RSA for card
- *Enhancements:*
  - encryption randomized by including random r: $m \leftarrow PIN; r$
  - for freshness: include challenge c from card $m \leftarrow PIN; r; c$

It is hard to get RSA encryption of data right

Using RSA for encryption: solutions

- Apply a hybrid scheme:
  - use RSA for encrypting a symmetric key K
  - encrypt (and authenticate) with symmetric cryptography
- Sending an encrypted key
  - randomize message before encryption
  - add redundancy and verify it after decryption
  - if NOK, return error

- The dominant standard is PKCS #1
- It specifies two versions: v1.5 and v2
  - v1.5 randomizes input but has no security proof
  - v2 is RSA-OAEP: randomizes input and uses hash function h
    - IND-CPA secure if inverting RSA is hard and the hash function is modeled as a random oracle ($\mathsf{h} \approx \mathcal{RO}$)
    - rather complex and hard to implement correctly
  - v1.5 most widespread

Key encapsulation with RSA

Hybrid encryption scheme using RSA-KEM:

Bob has Alice's public key (n, e)		Alice with private key (n, d)
$r \xleftarrow{\$} \mathbb{Z}/n\mathbb{Z}$		
$c \leftarrow r^e \bmod n$		
$K \leftarrow \mathrm{h}(\text{“KDF”}; r)$		
$CT \leftarrow \mathrm{Enc}_K(m)$	$\xrightarrow{c, CT}$	$r \leftarrow c^d \bmod n$
		$K \leftarrow \mathrm{h}(\text{“KDF”}; r)$
		$m \leftarrow \mathrm{Dec}_K(CT)$

- The *hybrid* encryption scheme including RSA-KEM is proven IND-CPA secure if
  - inverting RSA is hard
  - $\mathrm{h} \approx \mathcal{RO}$
  - the symmetric cryptosystem is secure
- Much simpler than RSA-OAEP
- RSA-KEM is the sound way to use RSA for exchanging a key

Problems of textbook RSA signatures

Textbook RSA signature:

Alice with private key (n, d)		Bob with Alice's public key (n, e)
$s \leftarrow m^d \bmod n$	$\xrightarrow{\text{Alice}, m, s}$	$m \stackrel{?}{=} s^e \bmod n$

Problems:

- RSA *malleability*
  - given signatures $s_1 = m_1^d$ and $s_2 = m_2^d$, Eve can sign $m_3 = m_1 \cdot m_2 \bmod n$ 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

Full-domain hash (FDH) RSA signature:

Alice with private key (n, d)		Bob with Alice's public key (n, e)
$H \leftarrow \mathsf{h}(m)$		
$s \leftarrow H^d \bmod n$	$\xrightarrow{\text{Alice},m,s}$	$H \leftarrow \mathsf{h}(m)$
		$H \stackrel{?}{=} s^e \bmod n$

- Secure against forgery if
  - inverting RSA is hard and
  - the hash function behaves like a random oracle ($\mathsf{h} \approx \mathcal{RO}$) ...
  - with co-domain of h equal to $\mathbb{Z}/n\mathbb{Z}$
  - this is called *full-domain hash*
- Can easily be realized by using XOF
  - generate output string longer than the length of n
  - interpret the result as an integer and reduce modulo n
- FDH did not make it to the standards (yet)

RSA signature standards

- Most widespread standards: PKCS # 1 v1.5 or v2 (RSA PSS)
  - First hashes message $H = \mathrm{h}(m)$ with classical hash function
  - then embeds H into the RSA input in $\mathbb{Z}/n\mathbb{Z}$ ...
  - ... uses padding and some messy processing
  - processing includes hash function calls to destroy malleability
  - used by the cool crowd of Silicon Valley
- Also widespread: ISO/IEC 9796-2
  - similar to PKCS # 1 but has a unique feature ...
  - ... *message recovery*
  - allows to stuff part of the signed message *inside the signature*
  - used in payment card standard EMV (not cool)

RSA vs ECC [for info only]

Computational efficiency of RSA [for info only]

- Public exponentiation is light (assuming $e = 2^{16} + 1$))
  - 16 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 [for info only]

- 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: $|q| = 256$ following general consensus including `keylength.com`
  - RSA: $|n| = 3072$ following advice on `keylength.com`

key lengths	RSA		ECC	
domain parameters	none		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 [for info only]

- 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
  - overhead can be reduced to about 256 bits

RSA-KEM vs ECDH [for info only]

- Computation
  - RSA-KEM: light on sending side and heavy on receiving
  - ECDH has same workload on both sides and is lighter
  - ECDH is much lighter on receiving end than RSA
  - 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 (compressed points): 770 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 establishment, signatures, and much more
- RSA is considered less *cool* than ECC but has unique advantages
  - faster encryption and signature verification
  - shorter signature overhead when using data recovery
- But actually, many applications can do without public key crypto
  - symmetric crypto may be sufficient
  - orders of magnitudes faster and 128-bit keys and tags
  - advice: study the requirements of the use case