

Cryptography: RSA and the discrete logarithm problem

R. Hayden

Advanced Maths Lectures
Department of Computing
Imperial College London

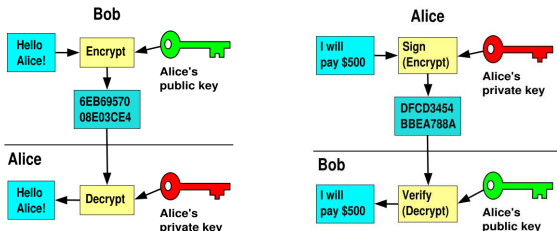
February 2010

Public key cryptography

Asymmetric cryptography — two keys:

- **Public key** — widely distributed
- **Private key** — users keep secret

Mathematically related, but cleartext (and thus private key) **hopefully** not practically computable given just public key



Trapdoor one-way functions

We need a function with the following properties:

- **Easy** to compute
- Inverse is **hard** to compute **without special information**
- **With special information**, inverse is also **easy** to compute

RSA

- First public key algorithm which also works for signing
- Discovered in 1973 by Clifford Cocks, mathematician working at GCHQ, UK intelligence agency. Top secret, only published internally, revealed in 1997
- First publicly described in 1977 by Ron Rivest, Adi Shamir and Leonard Adleman (independently discovered)

RSA

- First public key algorithm which also works for signing
- Discovered in 1973 by Clifford Cocks, mathematician working at GCHQ, UK intelligence agency. Top secret, only published internally, revealed in 1997
- First publicly described in 1977 by Ron Rivest, Adi Shamir and Leonard Adleman (independently discovered)

RSA

- First public key algorithm which also works for signing
- Discovered in 1973 by Clifford Cocks, mathematician working at GCHQ, UK intelligence agency. Top secret, only published internally, revealed in 1997
- First publicly described in 1977 by Ron **R**ivest, Adi **S**hamir and Leonard **A**dleman (independently discovered)

Key generation

- Each user chooses two large primes p and q and computes $n = pq$ and $\sigma = (p - 1)(q - 1)$
- Discard p and q
- Choose e and d such that $ed \equiv 1 \pmod{\sigma}$
- **Public key:** (e, n)
- **Private key:** d

Message representation

- Represent the message to be encrypted as an integer $m < n$, e.g. ASCII — interpret message as a number in base 256
 - Split the message into chunks if necessary — **usually encrypt a key for a symmetric algorithm**
- m must be coprime to n (we will see why). Only $p + q - 1$ numbers less than n *not coprime* to n :

$$1, p, 2p, \dots, (q-1)p, q, 2q, \dots, (p-1)q$$

Their proportion is:

$$\frac{p+q-1}{pq} \approx 1/p + 1/q$$

Can just add padding characters if necessary

Message representation

- Represent the message to be encrypted as an integer $m < n$, e.g. ASCII — interpret message as a number in base 256
 - Split the message into chunks if necessary — **usually encrypt a key for a symmetric algorithm**
- **m must be coprime to n** (we will see why). Only $p + q - 1$ numbers less than n *not coprime* to n :

$$1, p, 2p, \dots, (q-1)p, q, 2q, \dots, (p-1)q$$

Their proportion is:

$$\frac{p+q-1}{pq} \approx 1/p + 1/q$$

Can just add padding characters if necessary

Encryption and decryption

- If message is m , compute ciphertext c as:

$$c \equiv m^e \pmod{n}$$

- Message can be recovered (decrypted) by computing:

$$c^d \equiv m \pmod{n}$$

- Can a computer calculate modular exponents quickly? To find $a^b \pmod{c}$, expand b in base 2, e.g.

$$b = 1493 = 1024 + 256 + 128 + 64 + 16 + 4 + 1$$

And use $a^{2^k} \equiv (a^{2^{k-1}})^2 \pmod{c}$

Fermat's Little Theorem again

We need to show $c^d \equiv m \pmod n$.

Recall from last week:

Theorem (Fermat's Little Theorem)

$$a^{p-1} \equiv 1 \pmod p$$

for any prime p and integer $1 \leq a \leq p - 1$.

We proved this using the group \mathbb{Z}_p^\times

Why does decryption work?

We wanted to show $c^d \equiv m \pmod n$.

$$\begin{aligned} c^d &\equiv m^{ed} \pmod n && \text{by defn. of } c \\ &\equiv m^{1+k(p-1)(q-1)} \pmod n && \text{by defn. of } e, d \\ &\equiv m \cdot m^{k(p-1)(q-1)} \pmod{n = pq} \end{aligned}$$

Now apply FLT twice (recall we chose **m coprime to n**)

$$(m^{k(p-1)})^{q-1} \equiv 1 \pmod q \quad (m^{k(q-1)})^{p-1} \equiv 1 \pmod p$$

p and q are coprime, so $m^{k(p-1)(q-1)} \equiv 1 \pmod n$. Then:

$$c^d \equiv m \cdot 1 \equiv m \pmod n$$



Factoring will break RSA

Public key: (e, n) , private key: d

- If can factor $n = pq$, can break RSA
- Can then compute $\sigma = (p - 1)(q - 1)$
- Can find d satisfying $ed \equiv 1 \pmod{\sigma}$, i.e. $ed - 1 = k\sigma$ using Euclid's algorithm (cheap)

Factoring will break RSA

Public key: (e, n) , private key: d

- If can factor $n = pq$, can break RSA
- Can then compute $\sigma = (p - 1)(q - 1)$
- Can find d satisfying $ed \equiv 1 \pmod{\sigma}$, i.e. $ed - 1 = k\sigma$ using Euclid's algorithm (cheap)

Factoring will break RSA

Public key: (e, n) , private key: d

- If can factor $n = pq$, can break RSA
- Can then compute $\sigma = (p - 1)(q - 1)$
- Can find d satisfying $ed \equiv 1 \pmod{\sigma}$, i.e. $ed - 1 = k\sigma$ using Euclid's algorithm (cheap)

Factoring will break RSA

Public key: (e, n) , private key: d

- If can factor $n = pq$, can break RSA
- Can then compute $\sigma = (p - 1)(q - 1)$
- Can find d satisfying $ed \equiv 1 \pmod{\sigma}$, i.e. $ed - 1 = k\sigma$ using Euclid's algorithm (cheap)

Might be an easier way...

- Only need $\sigma = (p - 1)(q - 1)$ to break RSA. Might be an easier way than factoring to get σ ? **No!**
- If we know $\sigma = (p - 1)(q - 1)$, we know $pq - p - q + 1 = n - (p + q) + 1$ and thus we know $p + q$.
- Quadratic $x^2 - (p + q)x + pq = 0$ has roots p and q , which we can find using quadratic formula.

This is not a proof breaking RSA is as hard as factoring though...

Might be an easier way...

- Only need $\sigma = (p - 1)(q - 1)$ to break RSA. Might be an easier way than factoring to get σ ? **No!**
- If we know $\sigma = (p - 1)(q - 1)$, we know $pq - p - q + 1 = n - (p + q) + 1$ and thus we know $p + q$.
- Quadratic $x^2 - (p + q)x + pq = 0$ has roots p and q , which we can find using quadratic formula.

This is not a proof breaking RSA is as hard as factoring though...

Might be an easier way...

- Only need $\sigma = (p - 1)(q - 1)$ to break RSA. Might be an easier way than factoring to get σ ? **No!**
- If we know $\sigma = (p - 1)(q - 1)$, we know $pq - p - q + 1 = n - (p + q) + 1$ and thus we know $p + q$.
- Quadratic $x^2 - (p + q)x + pq = 0$ has roots p and q , which we can find using quadratic formula.

This is not a proof breaking RSA is as hard as factoring though...

Might be an easier way...

- Only need $\sigma = (p - 1)(q - 1)$ to break RSA. Might be an easier way than factoring to get σ ? **No!**
- If we know $\sigma = (p - 1)(q - 1)$, we know $pq - p - q + 1 = n - (p + q) + 1$ and thus we know $p + q$.
- Quadratic $x^2 - (p + q)x + pq = 0$ has roots p and q , which we can find using quadratic formula.

This is not a proof breaking RSA is as hard as factoring though...

Integer factorisation

- Some work suggests though that RSA **may be easier than factoring**. Nothing practical yet though...
- Recall that integer factorisation thought to have no polynomial time algorithm, **but not proven**
- RSA-640, \approx 30 years of single CPU time (5 calendar months actual). 4096-bit keys are the norm
- Sub-exponential integer factorisation algorithms do exist

Integer factorisation

- Some work suggests though that RSA **may be easier than factoring**. Nothing practical yet though...
- Recall that integer factorisation thought to have no polynomial time algorithm, **but not proven**
- RSA-640, \approx 30 years of single CPU time (5 calendar months actual). 4096-bit keys are the norm
- Sub-exponential integer factorisation algorithms do exist

Integer factorisation

- Some work suggests though that RSA **may be easier than factoring**. Nothing practical yet though...
- Recall that integer factorisation thought to have no polynomial time algorithm, **but not proven**
- RSA-640, \approx 30 years of single CPU time (5 calendar months actual). 4096-bit keys are the norm
- Sub-exponential integer factorisation algorithms do exist

Integer factorisation

- Some work suggests though that RSA **may be easier than factoring**. Nothing practical yet though...
- Recall that integer factorisation thought to have no polynomial time algorithm, **but not proven**
- RSA-640, \approx 30 years of single CPU time (5 calendar months actual). 4096-bit keys are the norm
- Sub-exponential integer factorisation algorithms do exist

Computational feasibility

- 4096-bit p and q . **Prime number theorem**, n th prime number, is $\approx n \ln n$. Max distance of random 4096-bit integer from a prime:

$$(n + 1) \ln(n + 1) - n \ln n \approx \ln(n + 1) \approx 4096 \cdot \ln 2 \approx 3000$$

Recall a **polynomial time** algorithm to **check primality**

- Finding e and d such that $ed \equiv 1 \pmod{\sigma}$ is an application of **Euclid's algorithm — polynomial time**
- Modular exponentiation is polynomial time** if we use base 2 idea from earlier
- So **RSA algorithm is computationally feasible**

Computational feasibility

- 4096-bit p and q . **Prime number theorem**, n th prime number, is $\approx n \ln n$. Max distance of random 4096-bit integer from a prime:

$$(n + 1) \ln(n + 1) - n \ln n \approx \ln(n + 1) \approx 4096 \cdot \ln 2 \approx 3000$$

Recall a **polynomial time** algorithm to **check primality**

- Finding e and d such that $ed \equiv 1 \pmod{\sigma}$ is an application of **Euclid's algorithm — polynomial time**
- Modular exponentiation is polynomial time** if we use base 2 idea from earlier
- So **RSA algorithm is computationally feasible**

Computational feasibility

- 4096-bit p and q . **Prime number theorem**, n th prime number, is $\approx n \ln n$. Max distance of random 4096-bit integer from a prime:

$$(n + 1) \ln(n + 1) - n \ln n \approx \ln(n + 1) \approx 4096 \cdot \ln 2 \approx 3000$$

Recall a **polynomial time** algorithm to **check primality**

- Finding e and d such that $ed \equiv 1 \pmod{\sigma}$ is an application of **Euclid's algorithm — polynomial time**
- **Modular exponentiation is polynomial time** if we use base 2 idea from earlier
- So **RSA algorithm is computationally feasible**

Computational feasibility

- 4096-bit p and q . **Prime number theorem**, n th prime number, is $\approx n \ln n$. Max distance of random 4096-bit integer from a prime:

$$(n + 1) \ln(n + 1) - n \ln n \approx \ln(n + 1) \approx 4096 \cdot \ln 2 \approx 3000$$

Recall a **polynomial time** algorithm to **check primality**

- Finding e and d such that $ed \equiv 1 \pmod{\sigma}$ is an application of **Euclid's algorithm — polynomial time**
- **Modular exponentiation is polynomial time** if we use base 2 idea from earlier
- So **RSA algorithm is computationally feasible**

Remember the group \mathbb{Z}_p^\times

$(\mathbb{Z}_p^\times, \times_p)$ is the multiplication table of $\{1, \dots, p-1\}$ under multiplication modulo p , e.g. $p=7$:

\times_7	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	1	3	5
3	3	6	2	5	1	4
4	4	1	5	2	6	3
5	5	3	1	6	4	2
6	6	5	4	3	2	1

Question: What is the least integer k such that $5^k = 2$?

Remember the group \mathbb{Z}_p^\times

$(\mathbb{Z}_p^\times, \times_p)$ is the multiplication table of $\{1, \dots, p-1\}$ under multiplication modulo p , e.g. $p=7$:

\times_7	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	1	3	5
3	3	6	2	5	1	4
4	4	1	5	2	6	3
5	5	3	1	6	4	2
6	6	5	4	3	2	1

Question: What is the least integer k such that $5^k = 2$?

Remember the group \mathbb{Z}_p^\times

$(\mathbb{Z}_p^\times, \times_p)$ is the multiplication table of $\{1, \dots, p-1\}$ under multiplication modulo p , e.g. $p=7$:

\times_7	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	1	3	5
3	3	6	2	5	1	4
4	4	1	5	2	6	3
5	5	3	1	6	4	2
6	6	5	4	3	2	1

Question: What is the least integer k such that $5^k = 2$?

Answer: $((5 \times_7 5 = 4) \times_7 5 = 6) \times_7 5 = 2$.

So $5^4 = 2$. Or, $\log_5(2) = 4$.

Key generation

- Generate an **efficient description of the group G** of order q with **generator g**
- Choose a **random $0 \leq k \leq q - 1$**
- Compute **$h = g^k$**
- Publish (G, q, g, h) as **public key**
- k is **private key**

Key generation

- Generate an **efficient description of the group G** of order q with **generator g**
- Choose a **random $0 \leq k \leq q - 1$**
- Compute $h = g^k$
- Publish (G, q, g, h) as **public key**
- k is **private key**

Key generation

- Generate an **efficient description of the group G** of order q with **generator g**
- Choose a **random $0 \leq k \leq q - 1$**
- Compute **$h = g^k$**
- Publish (G, q, g, h) as **public key**
- k is **private key**

Key generation

- Generate an **efficient description of the group G** of order q with **generator g**
- Choose a **random $0 \leq k \leq q - 1$**
- Compute **$h = g^k$**
- Publish (G, q, g, h) as **public key**
- k is **private key**

Key generation

- Generate an **efficient description of the group G** of order q with **generator g**
- Choose a **random $0 \leq k \leq q - 1$**
- Compute **$h = g^k$**
- Publish (G, q, g, h) as **public key**
- k is **private key**

Encryption/decryption

Encryption:

- Convert message m into an element of G
- Choose a random $0 \leq y \leq q - 1$, calculate $c_1 = g^y$ and $c_2 = mh^y$
- Transmit **ciphertext** (c_1, c_2)

Decryption:

- Compute:

$$(c_2)(c_1^k)^{-1} = mh^y(g^{ky})^{-1} = m(g^{ky})(g^{ky})^{-1} = m$$

Encryption/decryption

Encryption:

- Convert message m into an element of G
- Choose a random $0 \leq y \leq q - 1$, calculate $c_1 = g^y$ and $c_2 = mh^y$
- Transmit **ciphertext** (c_1, c_2)

Decryption:

- Compute:

$$(c_2)(c_1^k)^{-1} = mh^y(g^{ky})^{-1} = m(g^{ky})(g^{ky})^{-1} = m$$

Breaking ElGamal

- Solving DLP problem for group G lets you calculate k
- **Open question** whether breaking ElGamal is as hard as solving DLP in general
- In some cases, DLP is very easy
- For $G = \mathbb{Z}_p^\times$, **the DLP is (a bit) harder than factoring**
- But there is still a **sub-exponential algorithm** (*index calculus*)

Breaking ElGamal

- Solving DLP problem for group G lets you calculate k
- **Open question** whether breaking ElGamal is as hard as solving DLP in general
- In some cases, DLP is very easy
- For $G = \mathbb{Z}_p^\times$, **the DLP is (a bit) harder than factoring**
- But there is still a **sub-exponential algorithm** (*index calculus*)

Breaking ElGamal

- Solving DLP problem for group G lets you calculate k
- **Open question** whether breaking ElGamal is as hard as solving DLP in general
- In some cases, DLP is very easy
- For $G = \mathbb{Z}_p^\times$, **the DLP is (a bit) harder than factoring**
- But there is still a **sub-exponential algorithm** (*index calculus*)

Breaking ElGamal

- Solving DLP problem for group G lets you calculate k
- **Open question** whether breaking ElGamal is as hard as solving DLP in general
- In some cases, DLP is very easy
- For $G = \mathbb{Z}_p^\times$, **the DLP is (a bit) harder than factoring**
- But there is still a **sub-exponential algorithm** (*index calculus*)

Breaking ElGamal

- Solving DLP problem for group G lets you calculate k
- **Open question** whether breaking ElGamal is as hard as solving DLP in general
- In some cases, DLP is very easy
- For $G = \mathbb{Z}_p^\times$, **the DLP is (a bit) harder than factoring**
- But there is still a **sub-exponential algorithm** (*index calculus*)

Elliptic curve groups

- Realise that you can do ElGamal on **any cyclic group**
- **Elliptic curve groups** consist of the points (x, y) on curves of the form:

$$y^2 = x^3 + ax + b$$

- These points form a group under a binary operation with a **beautiful geometric interpretation**

Elliptic curve groups

- Realise that you can do ElGamal on **any cyclic group**
- **Elliptic curve groups** consist of the points (x, y) on curves of the form:

$$y^2 = x^3 + ax + b$$

- These points form a group under a binary operation with a **beautiful geometric interpretation**

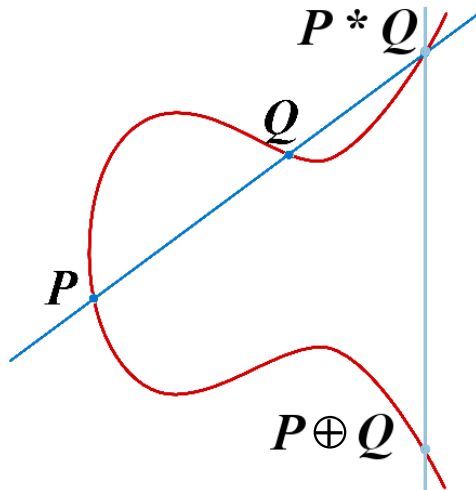
Elliptic curve groups

- Realise that you can do ElGamal on **any cyclic group**
- **Elliptic curve groups** consist of the points (x, y) on curves of the form:

$$y^2 = x^3 + ax + b$$

- These points form a group under a binary operation with a **beautiful geometric interpretation**

Elliptic curve group law



Elliptic curve cryptography

- Over finite fields, elliptic curve groups are finite
- So we can use DLP
- **No sub-exponential algorithm** exists
- This means ECC is probably **safer** and/or **more environmentally friendly** since we may use smaller key sizes
- For more on ECC and references, **see last year's ECC lecture** ([~ rh/files/uni/teaching/adv_maths/09/lecture3.pdf](http://rh/files/uni/teaching/adv_maths/09/lecture3.pdf))

Elliptic curve cryptography

- Over finite fields, elliptic curve groups are finite
- So we can use DLP
- **No sub-exponential algorithm** exists
- This means ECC is probably **safer** and/or **more environmentally friendly** since we may use smaller key sizes
- For more on ECC and references, **see last year's ECC lecture** ([~ rh/files/uni/teaching/adv_maths/09/lecture3.pdf](http://rh/files/uni/teaching/adv_maths/09/lecture3.pdf))

Elliptic curve cryptography

- Over finite fields, elliptic curve groups are finite
- So we can use DLP
- **No sub-exponential algorithm** exists
- This means ECC is probably **safer** and/or **more environmentally friendly** since we may use smaller key sizes
- For more on ECC and references, **see last year's ECC lecture** ([~ rh/files/uni/teaching/adv_maths/09/lecture3.pdf](http://rh/files/uni/teaching/adv_maths/09/lecture3.pdf))

Elliptic curve cryptography

- Over finite fields, elliptic curve groups are finite
- So we can use DLP
- **No sub-exponential algorithm** exists
- This means ECC is probably **safer** and/or **more environmentally friendly** since we may use smaller key sizes
- For more on ECC and references, **see last year's ECC lecture** ([~ rh/files/uni/teaching/adv_maths/09/lecture3.pdf](http://rh/files/uni/teaching/adv_maths/09/lecture3.pdf))

Elliptic curve cryptography

- Over finite fields, elliptic curve groups are finite
- So we can use DLP
- **No sub-exponential algorithm** exists
- This means ECC is probably **safer** and/or **more environmentally friendly** since we may use smaller key sizes
- For more on ECC and references, **see last year's ECC lecture** ([~ rh/files/uni/teaching/adv_maths/09/lecture3.pdf](http://rh/files/uni/teaching/adv_maths/09/lecture3.pdf))