Intro to Analysis of Algorithms
Randomized
Chapter 6

Michael Soltys

CSU Channel Islands

[Ed: 4th, last updated: September 16, 2025]

Primality testing

One way to determine whether a number p is prime, is to try all possible numbers $n < p$, and check if any are divisors.

This brute force procedure has exponential time complexity in the length of p, and so it has a prohibitive time cost.

Sieve of Eratosthenes

Eratosthenes of Cyrene
Born $\sim$ 276 BC and died $\sim$ 194 BC
Chief librarian at Library of Alexandria

Algorithm for finding all primes $\leq n$.

1: $L = \{2, 3, 4, \ldots, n\}$
2: $p = 2$
3: Mark all $2p, 3p, 4p, \ldots$ on list L
4: Find first unmarked number greater than p, say k.
5: Repeat with $p = k$

See simulation here:
`https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes`

Although a polytime (deterministic) algorithm for primality is now known[1], the Rabin-Miller *randomized algorithm* for primality testing is simpler and more efficient, and therefore still used in practice.

[1] Agrawal, Kayal and Saxena, August 2002.

Testing primality with `openssl version`:

OpenSSL 1.0.2g 1 Mar 2016

```
openssl prime 34891113111171233333111111111
```

Fermat's Little theorem provides a "test" of sorts for primality, called the Fermat test; the Rabin-Miller algorithm is based on this test. When we say that p passes the Fermat test at a, what we mean is that $a^{(p-1)} \equiv 1 \pmod{p}$. Thus, all primes pass the Fermat test for all $a \in \mathbb{Z}_p - \{0\}$.

Unfortunately, there are also composite numbers n that pass the Fermat tests for every $a \in \mathbb{Z}_n^*$; these are the so called *Carmichael numbers*, for example, 561, 1105, 1729, etc.

If p is a composite non-Carmichael number, then it passes at most half of the tests in $\mathbb{Z}_p^*$. That is, if p is a composite non-Carmichael number, then for at most half of the a's in the set $\mathbb{Z}_p^*$ it is the case that $a^{(p-1)} \equiv 1 \pmod{p}$.

Call a a *witness* (for p) if it fails the Fermat test for p, that is, a is a witness if $a^{(p-1)} \not\equiv 1 \pmod{p}$. Let $S \subseteq \mathbb{Z}_p^*$ consist of those elements $a \in \mathbb{Z}_p^*$ for which $a^{p-1} \equiv 1 \pmod{p}$. It is easy to check that S is in fact a subgroup of $\mathbb{Z}_p^*$.

Therefore, by Lagrange theorem $|S|$ must divide $|\mathbb{Z}_p^*|$. Suppose now that there exists an element $a \in \mathbb{Z}_p^*$ for which $a^{p-1} \not\equiv 1 \pmod{p}$. Then, $S \neq \mathbb{Z}_p^*$, so the next best thing it can be is "half" of $\mathbb{Z}_p^*$, so $|S|$ must be at most half of $|\mathbb{Z}_p^*|$.

A number is *pseudoprime* if it is either prime or Carmichael.

The last result suggests an algorithm for pseudoprimes: on input p, check whether $a^{(p-1)} \equiv 1 \pmod{p}$ for some random $a \in \mathbb{Z}_p - \{0\}$.

If p fails this test (i.e., $a^{(p-1)} \not\equiv 1 \pmod{p}$), then p is composite for sure.

If p passes the test, then p is probably pseudoprime.

We show that the probability of error in this case is $\leq \frac{1}{2}$.

Suppose p is not pseudoprime. If $\gcd(a, p) \neq 1$, then $a^{(p-1)} \not\equiv 1 \pmod{p}$, so assuming that p passed the test, it must be the case that $\gcd(a, p) = 1$, and so $a \in \mathbb{Z}_p^*$. But then at least half of the elements of $\mathbb{Z}_p^*$ are witnesses of non-pseudoprimeness.

Show that if $\gcd(a, p) \neq 1$ then $a^{(p-1)} \not\equiv 1 \pmod{p}$.

The informal algorithm for pseudoprimeness described in the paragraph above is the basis for the Rabin-Miller algorithm which we discuss next. The Rabin-Miller algorithm extends the pseudoprimeness test to deal with Carmichael numbers.

Rabin-Miller algorithm

1: If $n = 2$, accept; if n is even and $n > 2$, reject.
2: Choose at random a positive a in $\mathbb{Z}_n$.
3: **if** $a^{(n-1)} \not\equiv 1 \pmod{n}$ **then**
4: reject
5: **else**
6: Find s, h such that s is odd and $n - 1 = s2^h$
7: Compute the sequence $a^{s\cdot 2^0}, a^{s\cdot 2^1}, a^{s\cdot 2^2}, \ldots, a^{s\cdot 2^h} \pmod{n}$
8: **if** all elements in the sequence are 1 **then**
9: accept
10: **else if** the last element different from 1 is -1 **then**
11: accept
12: **else**
13: reject
14: **end if**
15: **end if**

Note that this is a polytime (randomized) algorithm.

If n is a prime then the Rabin-Miller algorithm accepts it; if n is composite, then the algorithm rejects it with probability $\geq \frac{1}{2}$.

If n is prime, then by Fermat's Little theorem $a^{(n-1)} \equiv 1 \pmod{n}$, so line 4 cannot reject n. Suppose that line 13 rejects n; then there exists a b in $\mathbb{Z}_n$ such that $b \not\equiv \pm 1 \pmod{n}$ and $b^2 \equiv 1 \pmod{n}$. Therefore, $b^2 - 1 \equiv 0 \pmod{n}$, and hence

$$(b-1)(b+1) \equiv 0 \pmod{n}.$$

Since $b \not\equiv \pm 1 \pmod{n}$, both $(b-1)$ and $(b+1)$ are strictly between 0 and n, and so a prime n cannot divide their product. This gives a contradiction, and therefore no such b exists, and so line 13 cannot reject n.

If n is an odd composite number, then we say that a is a *witness* (of compositness) for n if the algorithm rejects on a.

We show that if n is an odd composite number, then at least half of the a's in $\mathbb{Z}_n$ are witnesses.

The distribution of witnesses in $\mathbb{Z}_n$ appears to be very irregular, but if we choose our a "uniformly at random," we hit a witness with probability $\geq \frac{1}{2}$.

Because n is composite, either n is the power of an odd prime, or n is the product of two odd co-prime numbers.

This yields two cases.

Case 1. Suppose that $n = q^e$ where q is an odd prime and $e > 1$.

$$\text{Set } t := 1 + q^{e-1} \tag{1}$$

From the binomial expansion of t^n we obtain:

$$t^n = (1 + q^{e-1})^n = 1 + nq^{e-1} + \sum_{l=2}^{n} \binom{n}{l} (q^{e-1})^l, \tag{2}$$

$$\therefore t^n \equiv 1 \pmod{n} \tag{3}$$

- If $t^{n-1} \equiv 1 \pmod{n}$, then $t^n \equiv t \pmod{n}$ which contradicts (1) and (3).
  Hence t is a line 4 witness.
- But the set of line 4 non-witnesses,

$$S_1 := \{a \in \mathbb{Z}_n | a^{(n-1)} \equiv 1 \pmod{n}\},$$

  is a proper subgroup of $\mathbb{Z}_n^*$
  Note that $t \in \mathbb{Z}^* - S_1$
  by Lagrange's theorem S_1 is at most half of $\mathbb{Z}_n^*$, and so it is at most half of $\mathbb{Z}_n$.

Case 2. Suppose that $n = qr$, where q, r are co-prime.

Among all line 13 non-witnesses, find a non-witness for which the -1 appears in the largest position in the sequence in line 7 of the algorithm

Note that -1 is a line 13 non-witness, so the set of these non-witnesses is not empty.

Let x be such a non-witness and let j be the position of -1 in its sequence, where the positions are numbered starting at 0; $x^{s \cdot 2^j} \equiv -1 \pmod{n}$ and $x^{s \cdot 2^{j+1}} \equiv 1 \pmod{n}$.

The line 13 non-witnesses are a subset of

$$S_2 := \{a \in \mathbb{Z}_n^* | a^{s \cdot 2^j} \equiv \pm 1 \pmod{n}\},$$

and S_2 is a subgroup of $\mathbb{Z}_n^*$.

By the CRT there exists $t \in \mathbb{Z}_n$ such that

$$\begin{array}{ll} t \equiv x & (\text{mod } q) \\ t \equiv 1 & (\text{mod } r) \end{array} \quad \Rightarrow \quad \begin{array}{ll} t^{s \cdot 2^j} \equiv -1 & (\text{mod } q) \\ t^{s \cdot 2^j} \equiv 1 & (\text{mod } r) \end{array}$$

Hence t is a witness because $t^{s \cdot 2^j} \not\equiv \pm 1 \pmod{n}$ but on the other hand $t^{s \cdot 2^{j+1}} \equiv 1 \pmod{n}$.

Exercise: Show that $t^{s \cdot 2^j} \not\equiv \pm 1 \pmod{n}$.

Therefore, just as in case 1, we have constructed a $t \in \mathbb{Z}_n^*$ which is not in S_2, and so S_2 can be at most half of $\mathbb{Z}_n^*$, and so at least half of the elements in $\mathbb{Z}_n$ are witnesses.

Note that by running the algorithm k times on independently chosen a, we can make sure that it rejects a composite with probability $\geq 1 - \frac{1}{2^k}$ (it will always accept a prime with probability 1).

Thus, for $k = 100$ the probability of error, i.e., of a false positive, is *negligible*.

Summarize 3 important points

1. Rabin-Miller has 50% of a false-positive, but we can *amplify it away* by repeating the test to *negligable*.
2. As there are lots of primes ($\pi(n) \to n/\log(n)$) we can convert easily a tester of primes into a *generator of primes*.
3. For implementation we still have to show how to compute

$$g^x \pmod{n}$$

for large x, say in the order of 4K bits: *repeated squaring trick*.

Repeated Squaring

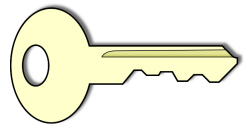
Computing large powers in $(\mathbb{Z}_n, *)$ can be done efficiently with *repeated squaring*—for example, if $(m)_b = c_r \dots c_1 c_0$, then compute

$$a_0 = a, a_1 = a_0^2, a_2 = a_1^2, \dots, a_r = a_{r-1}^2 \pmod{n},$$

and so $a^m = a_0^{c_0} a_1^{c_1} \cdots a_r^{c_r} \pmod{n}$.

Crypto

WEP, WPA/WPA2



SSL/SSH

PGP/GPG

amazon.com

```
RSA Encryption 128 bytes:
BE 89 0E A1 AD FA 7D 58 6A A1 6A E4
3B ED 75 E4 3E F2 19 F7 F3 0F FA D9
EF 62 10 52 7B FC DD 94 96 A8 35 6B
1B 50 60 2E 2E 79 AC 7C 2E A3 81 DE
8D 37 F9 EE 6E 4F 82 C7 E4 12 04 55
AF 57 69 94 8C EF 2E 50 7A 6D 53 0F
5B 5F 62 58 5E CF F2 DF F4 4D CE 71
B6 82 D7 86 E5 4F 77 E4 91 AA E4 BD
5A 65 AA 9E 20 4F 38 5E B4 8B E0 36
45 80 A8 D5 24 5C 46 9D F1 80 C0 6B
62 A5 1F 26 5E AE 17 47
```

DRM
FairPlay

MD5

```
5c3079df8a48623f5aa10f0181a7ab03
```

Cryptography is the art of *computing* & *communicating* in the presence of an adversary.

Cryptography is concerned with the conceptualization, definition, and construction of computing systems that address security concerns.

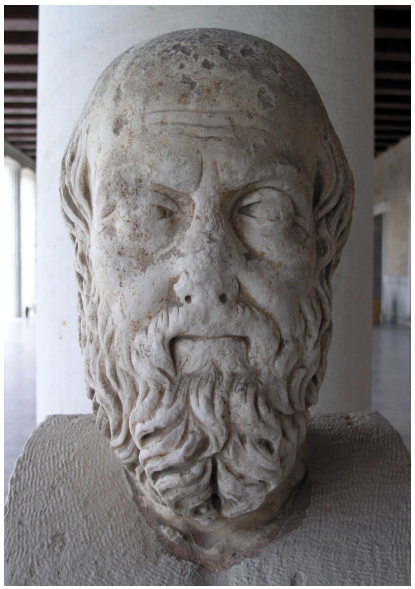
The word *cryptography* comes from the Greek word $\kappa\rho\upsilon\pi\tau o$ (hidden or secret) and $\gamma\rho\alpha\phi\eta$ (writing). It can be seen as the art of secret writing; this kind of crypto can provide services such as:

- *integrity checking*: recipient is reassured that the message was not altered;
- *authentication*: verifying someone's identity.

But also ...

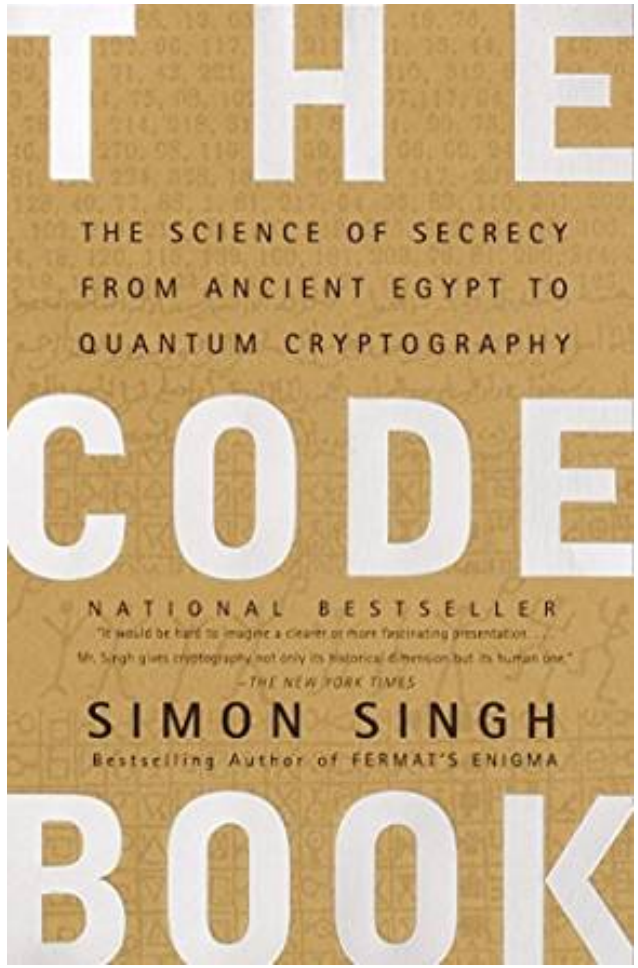
- Download a package and check its `MD5` signature; the adversary here is the "ether" through which the package "travels," e.g., *noisy channel*.
- Code "obfuscation" — compile code so that it is executable but cannot be reverse-engineered.
- Authentication: password login, client-server communication.
- Non-repudiation, electronic voting, digital cash, etc.

Origins



Herodotus chronicled the conflict between Greece & Persia, 5BC.

- Xerxes wanted to extend Persian empire; mobilizes a force.
- Persian military buildup witnessed by Demaratus. Writes message on wooden tablets, covers with wax and sends to Greece.
- Histaiaeus shaves the head of his messanger, writes the message on the scalp, and waits for hair to grow back.
- *Steganography*



Simon Singh, "The Code Book."

Enigma machine

http://enigmaco.de/enigma/enigma.html

Imitation Game (2014)

Enigma (2001)

Oxford (2005)

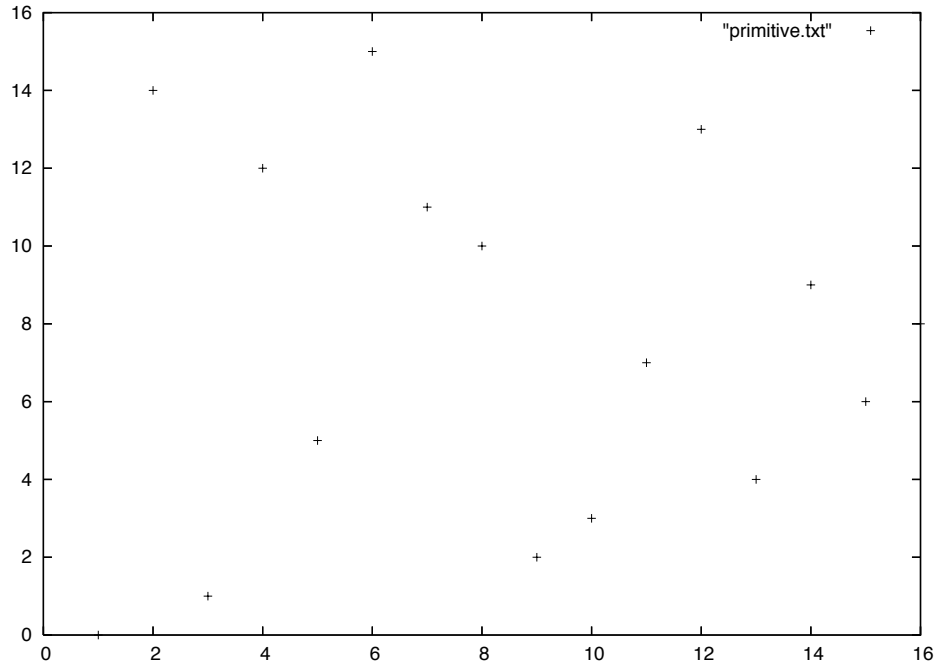
Oxford (2005)

Bletchley (2005)

The *discrete logarithm problem (DLP)*

Fix a group G and $g \in G$: given an element $h \in \langle g \rangle$, find an integer m satisfying $h = g^m$.

The smallest m satisfying $h = g^m$ is the *logarithm* (or *index*) of h with respect to g (written $m = \log_g(h)$).



Plot of $\log_3(x)$ over $\mathbb{Z}_{17}$.

Shank's "babystep-giantstep" algorithm

Computes x, such that $g^x \equiv_p h$, in time $O(\sqrt{p}\log(\sqrt{p}))$.

Pre-condition: p prime, $\langle g \rangle = \mathbb{Z}_p^*$, $h \in \mathbb{Z}_p^*$

1: $n \longleftarrow 1 + \lfloor \sqrt{p} \rfloor$
2: $L_1 \longleftarrow \{g^0, g^1, g^2, \ldots, g^n\} \pmod p$
3: $L_2 \longleftarrow \{hg^0, hg^{-n}, hg^{-2n}, \ldots, hg^{-n^2}\} \pmod p$
4: Find $g^i \equiv_p hg^{-jn} \in L_1 \cap L_2$
5: $x \longleftarrow jn + i$
6: **return** x

Post-condition: $g^x \equiv_p h$

PKCs

- Diffie-Hellman
- ElGamal
- RSA

Diffie-Hellman Key Exchange

- Oldest public key cryptosystem still in use.
- Allows two individuals to agree on a shared key, even though they can only exchange messages in public.
- A weakness is that there is no authentication; the other might be a "bad guy."
- Described in `RFC 2631`
  `https://www.ietf.org/rfc/rfc2631.txt`

Alice	Bob
Public: Group $G = \langle g \rangle$ and $n = \lvert G \rvert = \text{ord}(g)$	
Choose secret $0 < a < n$	Choose secret $0 < b < n$
Computer $A := g^a$	Compute $B := g^b$
Send A to Bob $\rightarrow$	$\leftarrow$ Send B to Alice
Compute B^a	Compute A^b
Alice & Bob have shared value $A^b = (g^a)^b = g^{ab} = g^{ba} = (g^b)^a = B^a$	

1. Alice and Bob agree to use a prime $p = 23$ and base $g = 5$.
2. Alice chooses secret $a = 8$; sends Bob $A = g^a \pmod p$
   - 2.1 $A = 5^8 \pmod{23}$
   - 2.2 $A = 16$
3. Bob chooses secret $b = 15$; sends Alice $B = g^b \pmod p$
   - 3.1 $B = 5^{15} \pmod{23}$
   - 3.2 $B = 19$
4. Alice computes $s = B^a \pmod p$
   - 4.1 $s = 19^8 \pmod{23}$
   - 4.2 $s = 9$
5. Bob computes $s = A^b \pmod p$
   - 5.1 $s = 16^{15} \pmod{23}$
   - 5.2 $s = 9$

The Diffie-Hellman Problem (DHP): compute g^{ab} from g^a and g^b.

If we could solve the DLP easily, we could solve the DHP easily.

For some groups DLP is easy: $(\mathbb{Z}_m, +)$, integers modulo m under addition, or $(\mathbb{R}^+, *)$, positive reals under multiplication.

For some groups, such as $(\mathbb{Z}_m, *)$, integers modulo m under multiplication, we believe that DLP is hard: the best known algorithm takes time:

$$O\left(e^{c\sqrt[3]{(\log m)(\log\log m)^2}}\right)$$

Diffie-Hellman enables two parties that have never communicated before to establish a mutual secret key by exchanging messages over a public channel. DH only resists *passive adversaries*.

A *passive attack* is one in which the intruder eavesdrops but does not modify the message stream in any way.

An *active attack* is one in which the intruder may:

- transmit messages
- replay old messages
- modify messages in transit
- delete selected messages from the wire

A typical active attack is one in which an intruder impersonates one end of the conversation, or acts as a *man-in-the-middle*.

How to do a "man-in-the-middle" on DH?

In DH it is recommended to make prime p 1024 bits long.

DH assumption: $k = g^{ab}$ hard to compute from g^a and g^b in $\mathbb{Z}_p^*$.

But, this does not guarantee that a particular bit or group of bits of k is also hard to compute.

Usually, the length of a symmetric session key is 128 bits.

Take the 128 most significant bits of $k = g^{ab} \pmod{p}$; call this $\tilde{k}$.

$\tilde{k}$ is still hard to compute from g^a and g^b.

In 1996 Boneh & Venkatesan show that computing $\sqrt{|p|}$ most significant bits is as hard as computing all the bits ($\sqrt{1024} = 32$).

ElGamal

Alice and Bob agree on public p, g, such that p is a prime and $\mathbb{Z}_p^* = \langle g \rangle$.

Alice also has a *private* a and publishes a *public* $A := g^a \pmod{p}$.

Bob wants to send a message m to Alice, so he creates an *ephemeral key* b, and sends the pair c_1, c_2 to Alice where:

$$c_1 := g^b \pmod{p}; \qquad c_2 := mA^b \pmod{p}.$$

Then, in order to read the message, Alice computes:

$$c_1^{-a} c_2 \equiv_p g^{-ab} m g^{ab} \equiv_p m.$$

To compute c_1^{-a} A computes the inverse of c_1 in $\mathbb{Z}_p^*$, using the extended Euclid's algorithm (EEA), and then compute the a-th power of the result.

To compute the inverse of some k in $\mathbb{Z}_n^*$, note first that $\gcd(k, n) = 1$.

So using EEA we obtain (*feasibly*!) s, t such that $sk + tn = 1$. Then the inverse is $s \pmod n$.

Pre-condition: $m > 0, n \geq 0$

1: $a \longleftarrow m$; $b \longleftarrow n$
2: **if** $b = 0$ **then**
3: **return** $(a, 1, 0)$
4: **else**
5: $(d, x, y) \longleftarrow \text{Euclid}(b, \text{rem}(a, b))$
6: **return** $(d, y, x - \text{div}(a, b) \cdot y)$
7: **end if**

Post-condition: $mx + ny = d = \gcd(m, n)$

Reductions

We say that we can *break* ElGamal, if we have an efficient way for computing m from $\langle p, g, A, c_1, c_2 \rangle$.

Reductions: We can break ElGamal if and only if we can solve the DHP efficiently.

The DHP on input $\langle p, g, A \equiv_p g^a, B \equiv_p g^b \rangle$ outputs $g^{ab} \pmod{p}$, and the ElGamal problem, call it ELGP, on input

$$\langle p, g, A \equiv_p g^a, c_1 \equiv_p g^b, c_2 \equiv_p mA^b \rangle \tag{4}$$

outputs m.

We want to show that we can break Diffie-Hellman, i.e., solve DHP efficiently, if and only if we can break ElGamal, i.e., solve ELGP efficiently.

The key-word here is *efficiently*, meaning in polynomial time.

($\Rightarrow$) Suppose we can solve DHP efficiently; we give an efficient procedure for solving ELGP: given the input (4) to ELGP, we obtain $g^{ab} \pmod p$ from $A \equiv_p g^a$ and $c_1 \equiv g^b$ using the efficient solver for DHP.

We then use the extended Euclidean algorithm to obtain $(g^{ab})^{-1} \pmod p$.

$$c_2 \cdot (g^{ab})^{-1} \equiv_p m g^{ab} (g^{ab})^{-1} \equiv_p m = m$$

where the last equality follows from $m \in \mathbb{Z}_p$.

($\Leftarrow$) Suppose we have an efficient solver for the ELGP. To solve the DHP, we construct the following input to ELGP:

$$\langle p, g, A \equiv_p g^a, c_1 \equiv_p g^b, c_2 = 1 \rangle.$$

Note that $c_2 = 1 \equiv_p \underbrace{(g^{ab})^{-1}}_{=m} A^b$, so using the efficient solver for ELGP we obtain $m \equiv_p (g^{ab})^{-1}$, and now using the extended Euclid's algorithm we obtain the inverse of $(g^{ab})^{-1} \pmod p$, which is just $g^{ab} \pmod p$, so we output that.

ElGamal signature scheme

We sign the hash value of message.

$\mathbb{Z}_p^* = \langle g \rangle$ and a is the secret key

k is random and $h : \mathbb{N} \longrightarrow \mathbb{Z}_p$ is a hash function

$1 < g, a, k < p - 1$, and $\gcd(k, p - 1) = 1$.

Compute:

$$r := g^k \pmod{p}$$
$$s := k^{-1}(h(m) - ar) \pmod{(p-1)}$$

If s is zero, start over again, by selecting a different k

The *signature of m* is the (ordered) pair of numbers (r, s).

For example, with the following values:

$$m = \texttt{A message.} \quad h(m) = 5 \quad p = 11 \quad g = 6 \quad a = 3 \quad k = 7$$

the signature of 'A message.' will be:

$$\begin{aligned} r &= 6^7 \pmod{11} = 8 \\ s &= 7^{-1}(5 - 3 \cdot 8) \pmod{(11-1)} \\ &= 3 \cdot (-19) \pmod{10} = 3 \cdot 1 \pmod{10} = 3 \end{aligned}$$

i.e., sign(A message.) $= (8, 3)$.

Suppose $h(m)$ is obtained by multiplying the ASCII values of the symbols in the message modulo 11.

Messages "A mess" and "L message" have the same hash value.

In general, by its nature, any hash function is going to have such *collisions*, i.e., messages such that:

$$h(\text{A message.}) = h(\text{A mess}) = h(\text{L message}) = 5,$$

but there are hash functions which are *collision-resistant* in the sense that it is computationally hard to find two messages m, m' such that $h(m) = h(m')$.

A good hash function is also a *one-way function* in the sense that given a value y it is computationally hard to find an m such that $h(m) = y$.

If you receive a message m, and a signature pair (r, s), and you only know p, g and $A = g^a \pmod{p}$, i.e., p, g, A are the *public* information, how can you "verify" the signature—and what does it mean to verify the signature?

Verifying the signature means checking that it was the person in possession of a that signed the document m.

Two subtle things: first we say "in possession of a" rather than the "legitimate owner of a," simply because a may have been compromised (e.g., stolen). This is not an *authentication* scheme!

Second, and this is why this scheme is so brilliant, we can check that "someone in possession of a" signed the message, even *without knowing what a is*!

We know A, where $A = g^a \pmod{p}$, but for large p, it is difficult to compute a from A (DLP).

Here is how we verify that "someone in possession of a" signed the message $m \in \mathbb{N}$.

First we check $0 < r < p$ and $0 < s < p - 1$.

Then we compute

$$v := g^{h(m)} \pmod{p}$$
$$w := A^r r^s \pmod{p}$$

Remember that g, p are public, m is known, and the function $h : \mathbb{N} \longrightarrow [p-1]$ is also known, and r, s is the given signature.

If $v = w$ then the signature is valid.

To see that this works note that we defined

$$\begin{aligned} s &:= k^{-1}(h(m) - ar) \pmod{p-1} \\ &\Rightarrow h(m) \equiv ar + sk \pmod{p-1} \qquad (*) \end{aligned}$$

Fermat's Little Theorem says that $g^{p-1} \equiv 1 \pmod{p}$, and therefore

$$g^{h(m)} \overset{(**)}{\equiv} g^{ar+sh} \equiv (g^a)^r (g^k)^s \equiv A^r r^s \pmod{p}.$$

The FLT is applied in the $(**)$ equality: from $(*)$ it follows that $(p-1)|(h(m) - (ar + sk))$, which means that $(p-1)z = h(m) - (ar + sk)$ for some z, and since $g^{(p-1)z} = (g^{(p-1)})^z = 1^z = 1 \pmod{p}$, it follows that $g^{h(m)-(ar+sk)} = 1 \pmod{p}$, and so $g^{h(m)} = g^{ar+sk} \pmod{p}$.

Without a (*good*) hash function, ElGamal's signature scheme is *existentially forgeable*; i.e., an adversary Eve can construct a message m and a valid signature (r, s) for m.

To see this, let b, c be numbers s.t. $\gcd(c, p-1) = 1$, and set

$$\begin{aligned} r &:= g^b A^c \\ s &:= -rc^{-1} \pmod{p-1} \\ m &:= -rbc^{-1} \pmod{p-1} \end{aligned}$$

Then (m, r, s) satisfies $g^m = A^r r^s$.

Since in practice a hash function h is applied to the message, and it is the hash value that is signed, to forge a signature for a meaningful message is not easy:

An adversary has to find a meaningful message m such that $h(m) \equiv -rbc^{-1} \pmod{p-1}$, which is hard.

In practice k is a random number; it is absolutely necessary to choose a *new* random number for each message.

If the same random number k is used in two different messages $m \neq m'$, then it is possible to compute k as follows:

$s - s' = (h(m) - h(m'))k^{-1} \pmod{p-1}$, and hence

$$k = (s - s')^{-1}(h(m) - h(m')) \pmod{p-1}$$

Once we have k, we can compute a.

In the verification it is essential to check that $1 \leq r \leq p-1$.

Otherwise E would be able to sign a message of his choice, provided he knows one valid signature (r, s) for some message m, where m is such that $1 \leq m \leq p-1$ and $\gcd(m, p-1) = 1$.

Let m' be a message of E's choice,

$$u \equiv m' m^{-1} \pmod{p-1}$$
$$s' \equiv su \pmod{p-1}$$

Let r' be any integer such that

$$r' \equiv r \pmod{p}$$
$$r' \equiv ru \pmod{p-1}$$

such an r' can be obtained by Chinese Reminder Theorem.

Exercise: Show (m', r', s') passes the verification.

RSA[2]

- The most commonly used key length for RSA is 1024 bits.
- The plaintext block must be smaller than the key.
- The ciphertext block will be the length of the key.
- RSA is much slower than secret key algorithms such as DES and IDEA.
- It is mostly used to secuerly exchange a secret key which is then used for symmetric key encryption.
- Security: to factor a 512-bit number takes, with today's technology, 30 thousand MIPS-years.
- RFC 2437

[2] Rivest-Shamir-Adleman

Choose two odd primes p, q, and set $n = pq$.

Choose $k \in \mathbb{Z}^*_{\phi(n)}$, $k > 1$.

Advertise f, where $f(m) \equiv m^k \pmod{n}$.

Compute l, the inverse of k in $\mathbb{Z}^*_{\phi(n)}$.

Now $\langle n, k \rangle$ are public, and the key l is secret, and so is the function g, where $g(C) \equiv C^l \pmod{n}$.

Note that $g(f(m)) \equiv_n m^{kl} \equiv_n m$.

Why $m^{kl} \equiv_n m$?

Observe that $kl = 1 + (-t)\phi(n)$, where $(-t) > 0$, and so $m^{kl} \equiv_n m^{1+(-t)\phi(n)} \equiv_n m \cdot \left(m^{\phi(n)}\right)^{(-t)} \equiv_n m$, because $m^{\phi(n)} \equiv_n 1$.

Note that this last statement does not follow directly from Euler's theorem because $m \in \mathbb{Z}_n$, and not necessarily in $\mathbb{Z}_n^*$.

To make sure that $m \in \mathbb{Z}_n^*$ it is enough to insist that we have $0 < m < \min\{p, q\}$; so we break a large message into small pieces.

It is interesting to note that we can bypass Euler's theorem, and just use Fermat's Little theorem:

We know that $m^{(p-1)} \equiv_p 1$ and $m^{(q-1)} \equiv_q 1$, so $m^{(p-1)(q-1)} \equiv_p 1$ and $m^{(q-1)(p-1)} \equiv_q 1$, thus $m^{\phi(n)} \equiv_p 1$ and $m^{\phi(n)} \equiv_q 1$.

This means that $p|(m^{\phi(n)} - 1)$ and $q|(m^{\phi(n)} - 1)$, so, since p, q are distinct primes, it follows that $(pq)|(m^{\phi(n)} - 1)$, and so $m^{\phi(n)} \equiv_n 1$.

Clifford Cocks, a British mathematician working for the UK intelligence agency GCHQ, described an equivalent system in an internal document in 1973, but given the relatively expensive computers needed to implement it at the time, it was mostly considered a curiosity and, as far as is publicly known, was never deployed.

His discovery, however, was not revealed until 1997 due to its top-secret classification, and Rivest, Shamir, and Adleman devised RSA independently of Cocks's work.