ONE-WAY FUNCTIONS

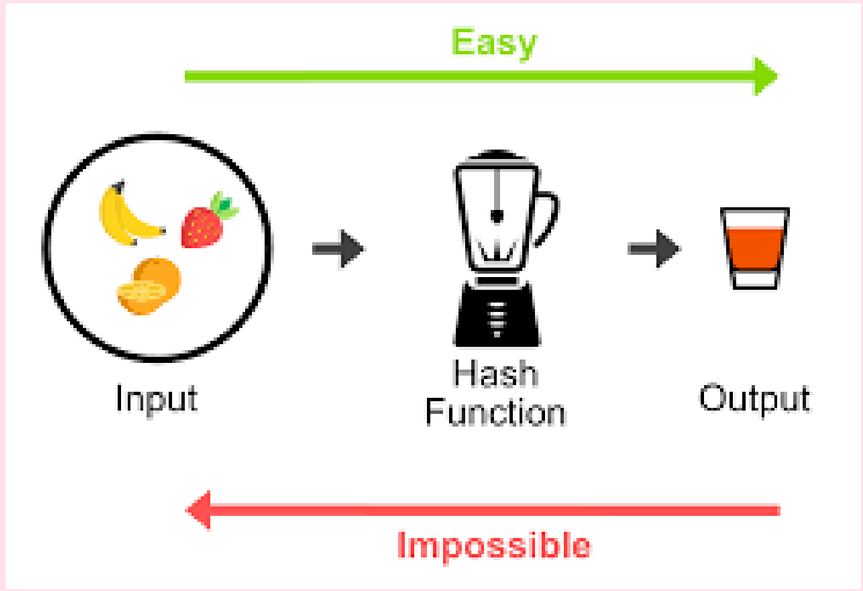
Matei Mara Sandra
Porof Ioana Gabriela

Faculty of Mathematics and Computer Science

:3

WHAT ARE ONE-WAY FUNCTIONS?

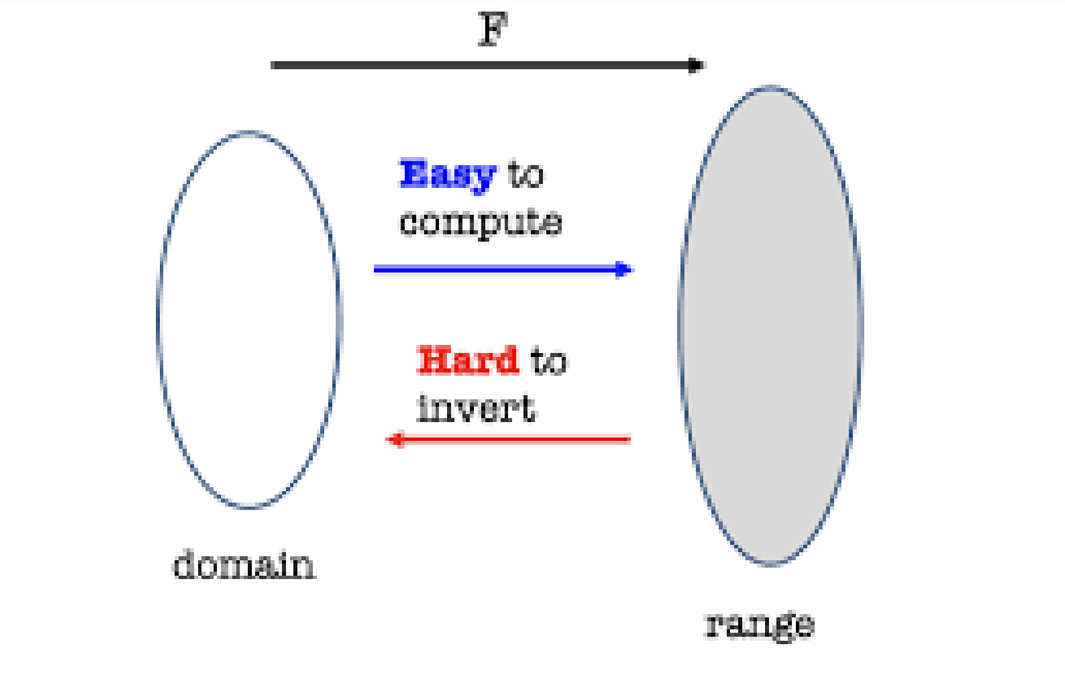
The existence of one-way functions is on the list of unsolved problems in mathematics and computer science. The deal with these functions is that they are easy to compute, but hard to invert.



Consider the following scenario: I make a smoothie and I put an amount of fruits of which I am aware of into it. Then, I give it to you and ask you to tell me the exact amount of each fruit used to make the drink. Would you be able to?

OR, ANOTHER EXAMPLE...

(and also a more formal definition)



I have a function F. If given x, I can easily compute F(x)=y, but, if given just y, it is virtually impossible for me to find out the exact x in order for F(x)=y.

So, what’s the point of these functions? Why would they interest us?

So, what’s the point of these functions? Why would they interest us?

One word: Cryptography

So, what's the point of these functions? Why would they interest us?

One word: Cryptography

If scientists could find a way to secure passwords or data, so that only those who hold the original key can access them, then the way we go about cryptography would be revolutionized.

So, why haven’t they? Having such functions would secure and protect a lot of vulnerable or valuable information.

Well, there's some bad news: the existence of one-way functions has not yet been proven. We only have candidate one-way functions, meaning, certain functions that we *believe* are one-way.

The following slides will talk about several such candidates, and why their efficiency in protecting data is widely acknowledged.

PRIME FACTORIZATION

Let’s say we have two distinct prime numbers. For the sake of this example, I have chosen two small ones:

x = 1931

y = 3571

Their product is easy to compute. Modern calculators do it in milliseconds (0.214 s). But, given the product x*y, in this case 6.895.601, and asked to find the x and the y? It's no longer a piece of cake.

Those two were rather small numbers. Numbers such as

359334085968622831041960188598043661065388726959079837

and

393050634124102232869567034555427371542904833

whose product is 98 digits long and looks like this:

998575172011145024715196447172945188768432256747176195623718008071534754622441555020697793335757 35

can become a problem.

In 1991, RSA Laboratories published a list of increasingly large numbers. The challenge? Find the two primes whose product is equal to each of the numbers on the list. There are 54 numbers on that list, of which several remain unsolved. The largest number has 617 digits (1024 bits) and here's how it looks:

25195908475657893494027183240048398571429282126204032027771378360
43662020707595556264018525880784406918290641249515082189298559149
17618450280848912007284499268739280728777673597141834727026189675 01
49718246911650776133798590957000973304597488084284017974291006424 5
86918171951187461215151726546322822168699875491824224336372590851418 6
54620435767984233871847744479207399342365848238242811981638150106 7
48104516603773060562016196762561338441436038339044149526344321901 14
65754445417842402092461651572335077870774981712577246796292638635 63
73289912154831438167899885040445364023527381951378636564391212010 3
9712282212072035 7

(and, yes, i needed a whole page in order for it to fit)

You would be right to doubt that such an operation is actually one-way. If, until now, computers struggled when presented with divisions, a new invention has emerged in latter years. Quantum computers, specifically built in order to solve complex operations.

Cryptography will need to find another way to encrypt and protect data.

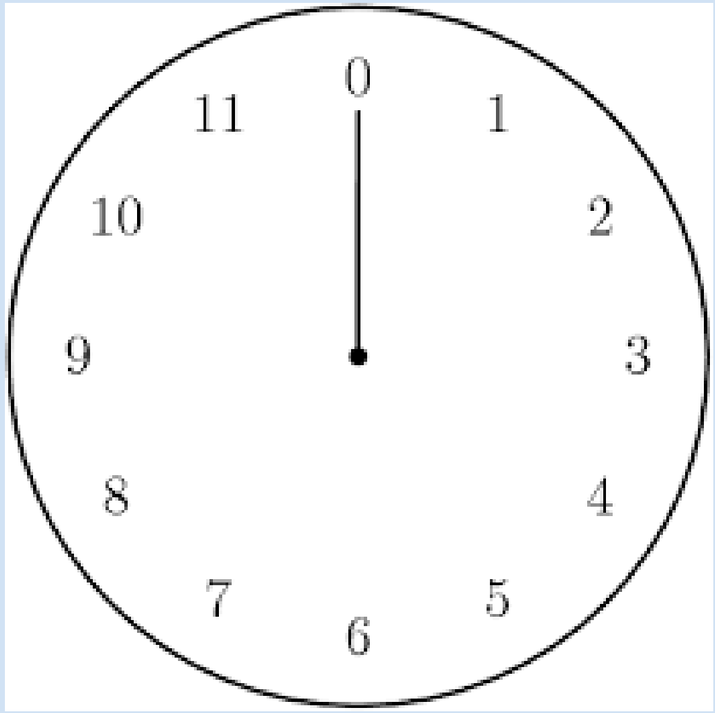
Another option?

Discrete Logarithm Problem

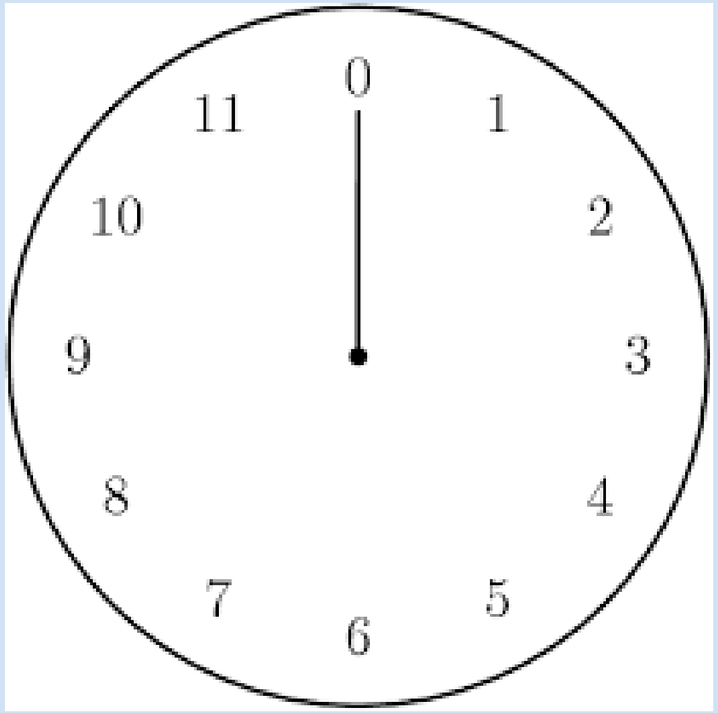
Given $b^k = a$, $k = \log_b a$, where k is an integer. But how could a simple logarithm be an one-way function and how can it be used in the field of cryptography?

Similarly to the Prime Factorization Problem, because computers hate divisions.

Let’s talk about modular algebra.

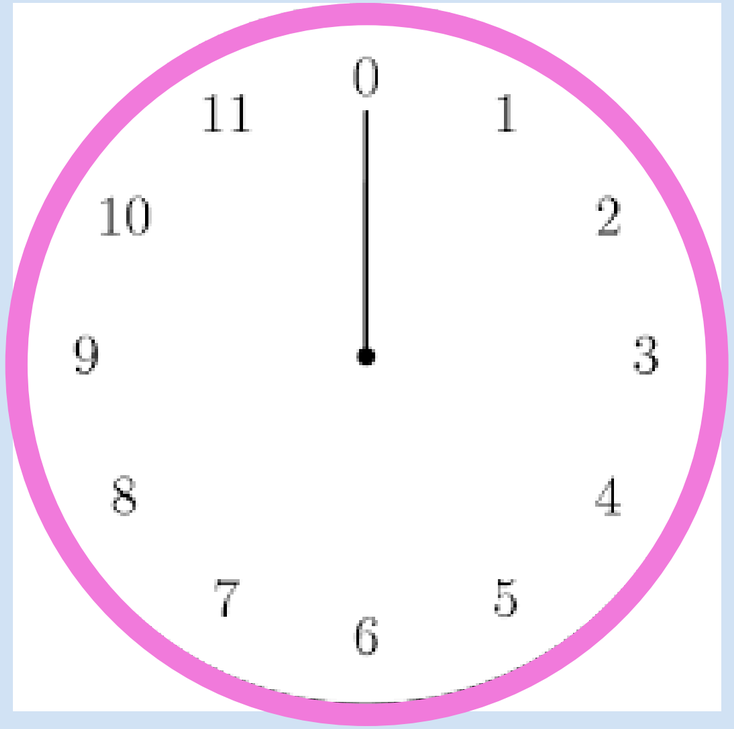


This is our clock → its length is 12 units



This is our clock → its length is 12 units

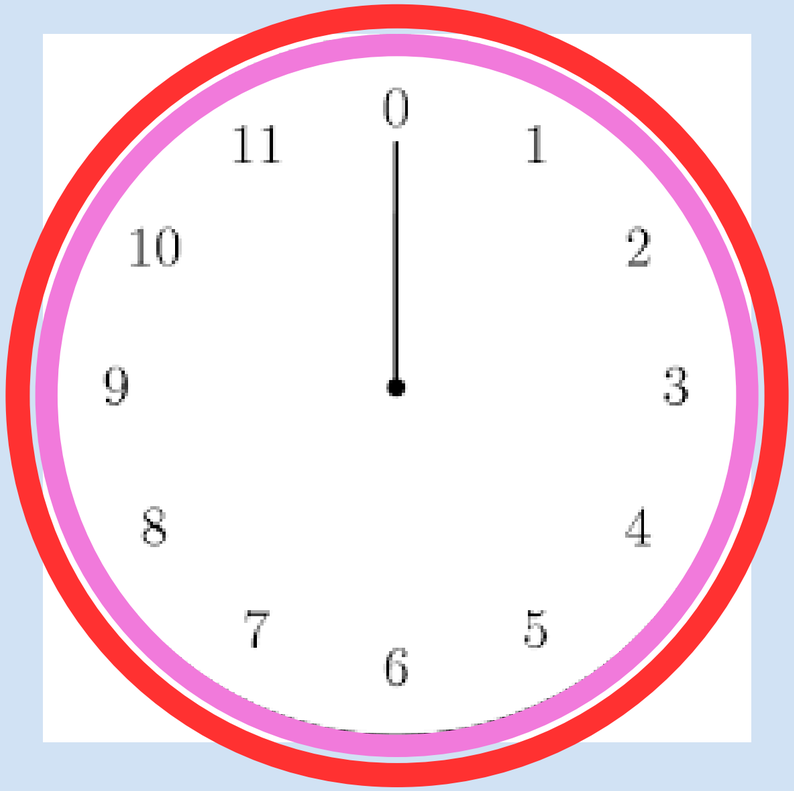
We now take a rope of 26 units, and wrap it around the clock



This is our clock → its length is 12 units

We now take a rope of 26 units, and wrap it around the clock

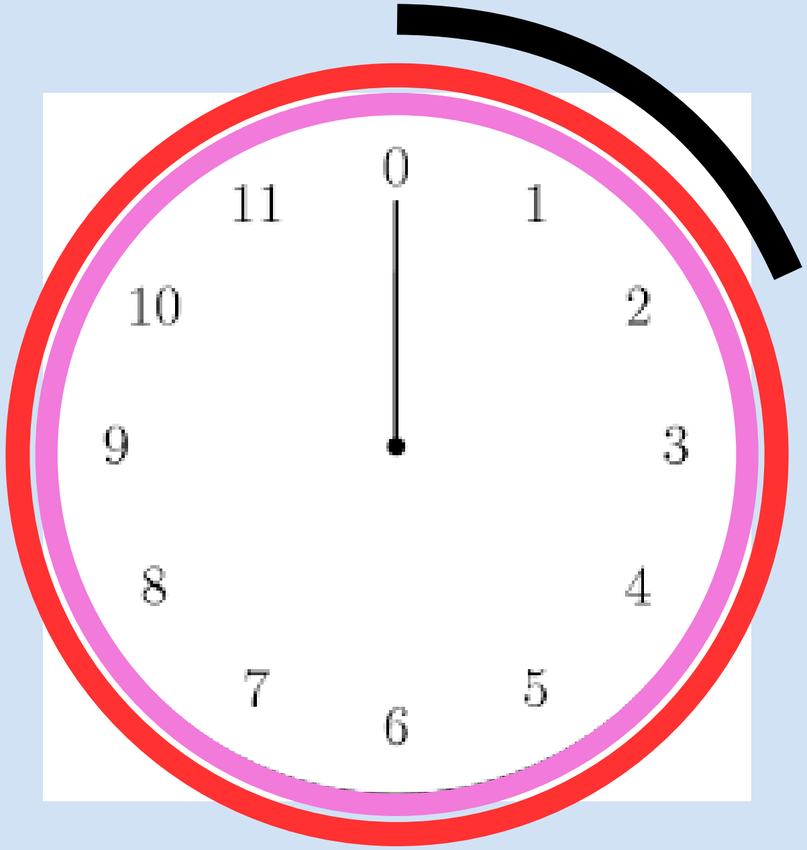
One time, we now have 26-12=14 units left of our rope



This is our clock → its length is 12 units

We now take a rope of 26 units, and wrap it around the clock

One time, we now have 26-12=14 units left of our rope



This is our clock → its length is 12 units

We now take a rope of 26 units, and wrap it around the clock

One time, we now have 26-12=14 units left of our rope

The second time, and now we have only 2 units lefts, which will bring us here → at unit 2

We now take a prime number as a modulus, like 13 or 17 or, even better, an abstract number p, and then we find a primitive root of that number.

What is a primitive root?

If you have a number k, raised to different powers, apply modulo n for each value and the results take all numbers from 0 to p-1, then that number k is a called a primitive root of p.

Let’s take an example:

let n = 17 and k = 3.

$3^0 = 1 \rightarrow 1 \% 17 = 1$

$3^1 = 3 \rightarrow 3 \% 17 = 3$

$3^2 = 9 \rightarrow 9 \% 17 = 9$

$3^3 = 27 \rightarrow 27 \% 17 = 10$

$3^4 = 81 \rightarrow 81 \% 17 = 13$

$3^5 = 243 \rightarrow 243 \% 17 = 5$

$3^6 = 729 \rightarrow 729 \% 17 = 15$

$3^7 = 2187 \rightarrow 2187 \% 17 = 11$

$3^8 = 6561 \rightarrow 6561 \% 17 = 16$

$3^9 = 19683 \rightarrow 19683 \% 17 = 14$

$3^{10} = 59049 \rightarrow 59049 \% 17 = 8$

$3^{11} = 177147 \rightarrow 177147 \% 17 = 7$

$3^{12} = 531441 \rightarrow 531441 \% 17 = 4$

$3^{13} = 1594323 \rightarrow 1594323 \% 17 = 12$

$3^{14} = 4782969 \rightarrow 4782969 \% 17 = 2$

$3^{15} = 14348907 \rightarrow 14348907 \% 17 = 6$

$3^{16} = 43046721 \rightarrow 43046721 \% 17 = 1$

Let's take an example:

let n = 17 and k = 3.

$3^0 = 1 \rightarrow 1 \% 17 = 1$

$3^1 = 3 \rightarrow 3 \% 17 = 3$

$3^2 = 9 \rightarrow 9 \% 17 = 9$

$3^3 = 27 \rightarrow 27 \% 17 = 10$

$3^4 = 81 \rightarrow 81 \% 17 = 13$

$3^5 = 243 \rightarrow 243 \% 17 = 5$

$3^6 = 729 \rightarrow 729 \% 17 = 15$

$3^7 = 2187 \rightarrow 2187 \% 17 = 11$

$3^8 = 6561 \rightarrow 6561 \% 17 = 16$

In red, we can see that all values between 1 and n-1 (16) are present.

This means that 3 is a primitive root modulo 17 and is known as the generator.

$3^9 = 19683 \rightarrow 19683 \% 17 = 14$

$3^{10} = 59049 \rightarrow 59049 \% 17 = 8$

$3^{11} = 177147 \rightarrow 177147 \% 17 = 7$

$3^{12} = 531441 \rightarrow 531441 \% 17 = 4$

$3^{13} = 1594323 \rightarrow 1594323 \% 17 = 12$

$3^{14} = 4782969 \rightarrow 4782969 \% 17 = 2$

$3^{15} = 14348907 \rightarrow 14348907 \% 17 = 6$

$3^{16} = 43046721 \rightarrow 43046721 \% 17 = 1$

This means that, when assigning values to x for:

$3^x \bmod 17$

the expression is equally likely to be *any* integer from 1 to 16.

As with previous examples, while it is easy to compute $3^x \bmod 17$, the reverse not so much.

Let's say, for

$3^x \bmod 17 = 5$, find x.

This is called the discrete logarithm problem.

We now have our one-way function candidate.

Sure, for small numbers like 3 and 17, it's doable even without the aid of a computer, but, for larger numbers, hundred of digits long, it not only becomes impractical, but even impossible for normal present technology.

Let’s take numbers that could be used in real life. Instead of 17, we have this prime number:

P = FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D
C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F
83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA237327 FFFFFFFF FFFFFFFF

Even though written in hexadecimal, it barely fits on one slide, so we can only imagine how its decimal version looks like.

The generator for this prime number is 2 and the equation looks like this:
$2^x \bmod P = Q$, where Q is a random number from 1 to P-1.
It is easy to find Q, when we know x, but the reverse is incredibly hard. And, even if we succeed in finding an x that satisfies the aforementioned condition, who is to say that it is the correct value? One of the most important aspects of modular arithmetic is that it is periodic. That means that there are multiple x that satisfy the equation and there is no way for us to know which one is the correct one unless we actually get it right.

While discussing the previous two candidates, we've seen that quantum computers are a growing threat for cryptography based on prime number factorization, but could the same be said about discrete logarithm problems?

Naturally, yes. This type of data encryption poses no issues for quantum computers. They can bypass it in polynomial time, using Shor’s Algorithm, which we will touch upon a bit later.

HASH FUNCTIONS

A hash-function takes an input and turns it into a string of fixed length. It has the following properties:

1. The probability of a particular n-bit output result for a random input string is 2^{-n} , so the hash value can be used as some sort of “fingerprint” for the input message, meaning it is always unique.
2. Finding the input message, when you only know the output is almost always impossible.

SHA-256 is part of the SHA-2 family of secure hash algorithms developed by the NSA in 2001. The members of the family are all named after the number of bits in their respective outputs (in this case, of course, 256).

Like every other hash function, SHA-256 is deterministic (a particular input will always generate the same output).

Consider the following example:

Using SHA-256 for these two different words:

1. “fugim": 2445f7880519842ea4433087df195be72849d115277564df11eebaf62468b05b
2. “fugem”: 5257adcdd2960a6e3e6cb9eb6853f7fd1a7ff9420b2925fa8357dabf4417168e

To further illustrate the point, here's another example:

1. “1”: 6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b
2. “0”: 5feceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9

One single flipped bit, and 2 resulting strings with no sequence of letters of significant length in common.

Hashes from the same function are of the same length, no matter the input. You couldn't tell an entire book and the number "8" apart.

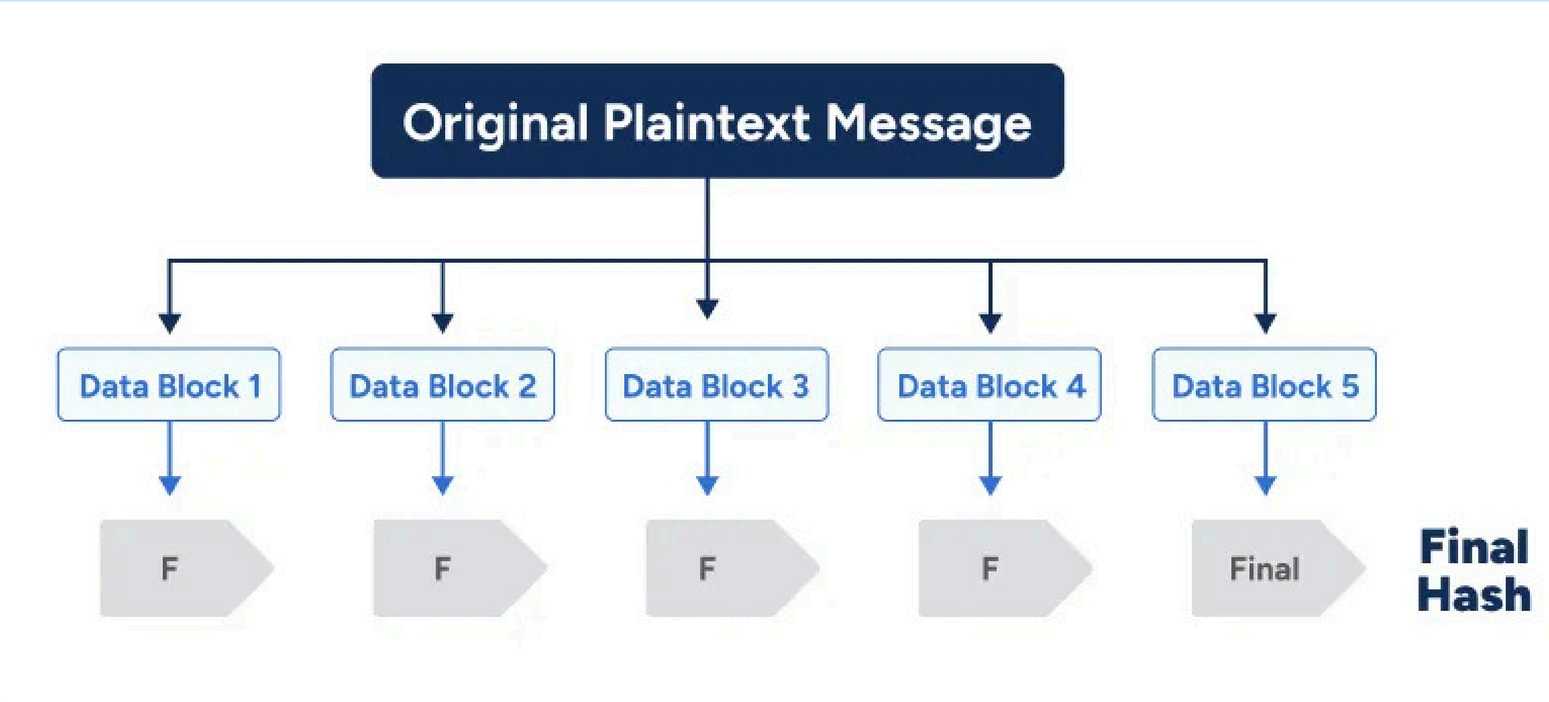
“But may God shield and deliver me from the fangs of the Arch-Fiend! No sooner had the reverberation of my blows sunk into silence, than I was answered by a voice from within the tomb!—by a cry, at first muffled and broken, like the sobbing of a child, and then quickly swelling into one long, loud, and continuous scream, utterly anomalous and inhuman—a howl—a wailing shriek, half of horror and half of triumph, such as might have arisen only out of hell, conjointly from the throats of the dammed in their agony and of the demons that exult in the damnation.”
(Edgar Allan Poe - “The black cat”)

faff6dd8e41f0adf3675b5c3939c6a29420c56c1699b3a238ad64f4b7dbb87e

“It had bewildered her, back at Canaan House, how the whole of her always seemed to come back to Gideon. For one brief and beautiful space of time, she had welcomed it: that microcosm of eternity between forgiveness and the slow, uncomprehending agony of the fall. Gideon rolling up her shirt sleeves. Gideon dappled in shadow, breaking promises. One idiot with a sword and an asymmetrical smile had proved to be Harrow’s end: her apocalypse swifter than the death of the Emperor and the sun with him.”
(Tamsyn Muir - “Harrow the Ninth”)

093dd0fa8f59814d51f64733e11b250937ad3a6d833ee51550122953cded9208

The algorithm divides the input equally into blocks and then the blocks processed before are used as input for the blocks to come.



After computing the first block, the algorithm uses the second block and also the output of the first block in order to compute the output for the second one.

The third block uses the outputs of the first and second blocks and so on.

The avalanche effect is a desirable property of hash functions, which ensures that if you modify the input however slightly (by a single bit, even), then the output drastically and unpredictably differs.

This helps to ensure that the data you are working with has not been tampered with beforehand.

The Strict Avalanche Criterion and the Bit Independence Criterion are popular criteria by which to improve this.

SAC

The flipping of a single bit in the input should result in each output bit having a 50% chance of changing.

BIC

The flipping of one bit in the input should affect each output bit independently of each other.

If the output for one input is the same as the output for a different input, the incident is called a “hash collision”. Hash collisions are unavoidable.

Though there are strategies for dealing with them (and thus rendering functions “collision resistant”, with an upper bound placed by the birthday paradox), they make the data vulnerable to brute-force attacks.

If the input is plain, there is little protection that can be offered.

Let’s take a look at the following table:

Number of characters	Numbers only	Lowercase only	Upper and lower	Number, upper and lower	Number, upper, lower and symbols
6	instantly	instantly	instantly	instantly	instantly
7	instantly	instantly	instantly	instantly	14 minutes
8	instantly	instantly	11 minutes	41 minutes	21 hours
9	instantly	instantly	9 hours	2 days	3 months
10	instantly	27 minutes	19 days	3 months	22 years
11	instantly	12 hours	2 years	19 years	2052 years
12	instantly	13 days	141 years	1164 years	195.000 years
13	2 days	9 months	7332 years	73.000 years	$19x10^6$ years
14	14 days	24 years	381.000 years	$4474x10^3$ years	$1760x10^6$ years

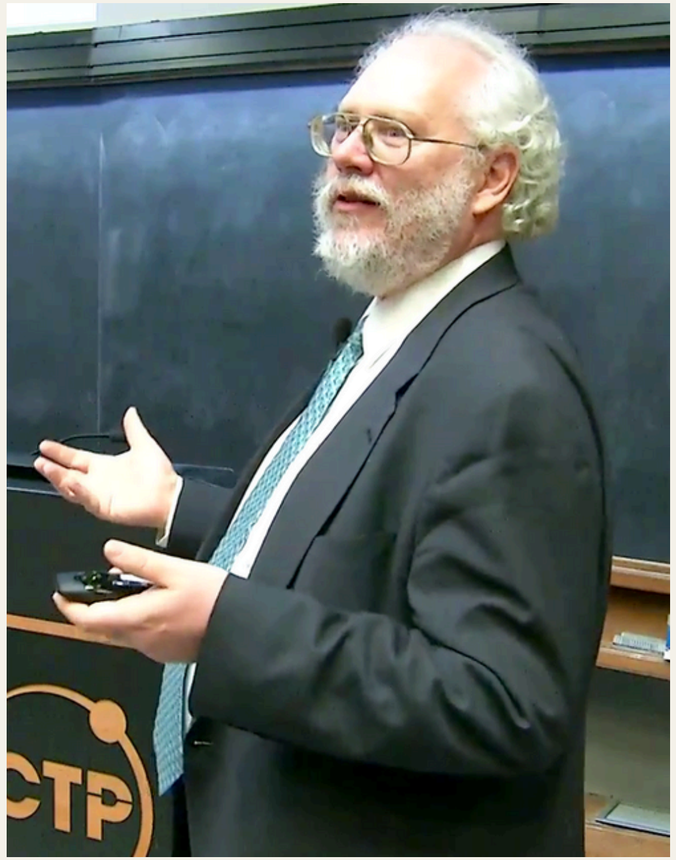
Number of characters	Numbers only	Lowercase only	Upper and lower	Number, upper and lower	Number, upper, lower and symbols
15	4 hours	605 years	$19x10^6$ years	$277x10^6$ years	$167.2x10^9$ years
16	2 days	15732 years	$1031x10^6$ years	$18x10^9$ years	16t years
17	14 days	$410x10^3$ years	$54x10^9$ years	$1067x10^9$ years	1509t years
18	5 months	$11x10^6$ years	$2788x10^9$ years	67t years	144q years
19	4 years	$277x10^6$ years	145t years	4099t years	14Q years
20	37 years	$7189x10^6$ years	8q years	255q years	1294Q years

If you want to steal something, you probably don’t feel like waiting around for a couple hundred years. Of course, this only applies if the password is not compromised. If it is, then all of these values turn to “instantly”.

But why do we need such algorithms? Why are quantic computers an exceptionally large threat?

But why do we need such algorithms? Why are quantic computers an exceptionally large threat?

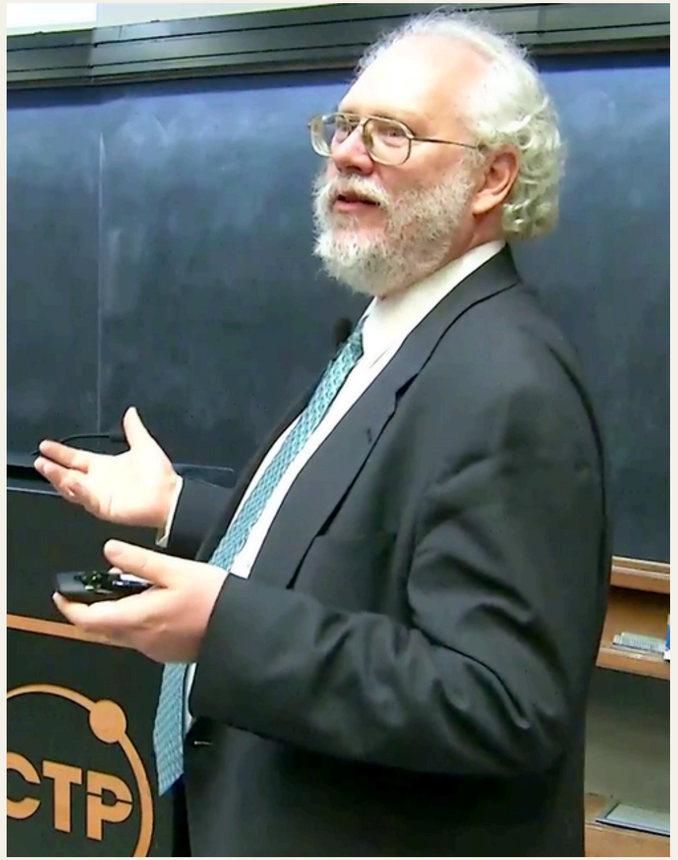
Because of this guy:



But why do we need such algorithms? Why are quantic computers an exceptionally large threat?

Because of this guy:

His name is Peter Shor, a theoretical computer scientist at MIT, known for devising Shor’s algorithm in 1994, which allows quantum computers to divide (factor) numbers exponentially faster than the best currently known algorithm.



SHOR'S ALGORITHM

Shor's algorithm is a way to solve the prime number factoring problem, discrete logarithm problem, and the period-finding problem in polynomial time, more precisely:

$$O((\log N)^2 (\log \log N)(\log \log \log N))$$

OR

$$O((\log N)^2 (\log \log N))$$

But...

how does the algorithm work?

Next come the steps for a successful implementation.

We will assume that our number N is odd and the product of two prime numbers.

We want to find the numbers that multiplied are equal to N.

For the sake of this explanation, let N=15

1. Pick a random number k, fulfilling the following condition:

$1 < k < N$

Let's say, k=11

2. Let D be the greatest common divisor of x and N.

3. If D is not 1, then D is a nontrivial factor of N and the other number is N/D and the problem is solved!! (unfortunately for us, D=1)

4. Now, we need to find the smallest natural number r such that if $f(x)=k^x \bmod N$, then $f(x)=f(x+r)$. Remember the discrete logarithm problem and clock arithmetic? We need to find the period of $k^x \bmod N$.

But, for a simpler explanation, let's break down this step into smaller ones:

4.1 We take a new variable q=1

4.2 We find (q x k)mod N. If this is equal to 1, great! If not, replace q with the new number we just found and repeat this step until we get the number 1. We count the steps and let r=nr of steps.

For our example, we have:

(11 x 1) mod 15 = 11

(11 x 11) mod 15 = 1

We count the steps, and we get r=2!

5. If r is odd, we turn back to step 2 and choose another value for k.

6. Let p = remainder of the (r/2)th transformation.

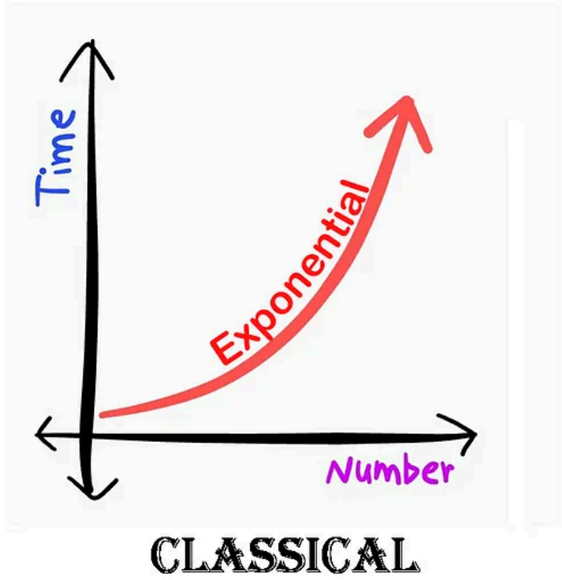
For us, the (r/2)th transformation is 2/2=1, so the first one. We take the remainder of this transformation, so p=11.

If p+1=N, then we return to step 2 and find another value for k.
For us, p+1=12 and it is not equal to 15. We can proceed.

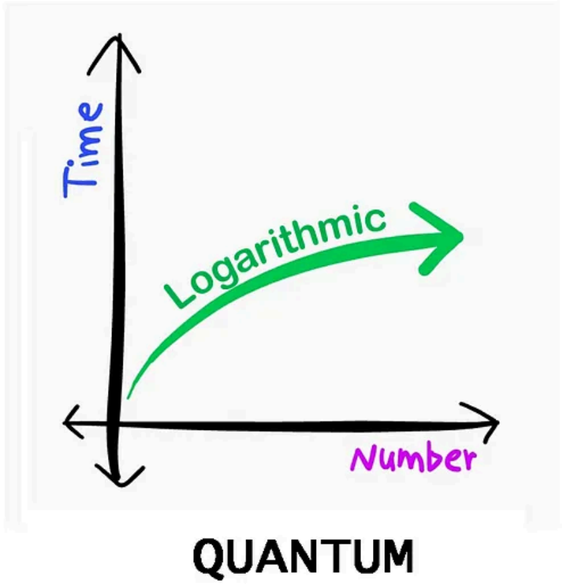
7. This is the final step. The factors of N are
GCD (greatest common divisor) of
(p+1, N) and (p-1, N).
So, we have GCD(12, 15)=3 and GCD(10,15)=5. Here we go, we have our two prime factors.

It may seem quite simple, but we must remember that this algorithm works only on quantum computers, due to step 4, which looks a lot like the discrete logarithm problem from earlier. So, for classical computers, solving a hard problem using the result of another hard problem is a vicious circle.

This algorithm is really efficient. It makes computing such numbers exponentially easier.



CLASSICAL



QUANTUM

There is just one problem, though. Shor was ahead of his time, that is for sure, but it looks like he is ahead of our time too.

For the quantum computers we have now, it is also physically impossible to apply this algorithm for a number bigger than 21. The reason? We have too few quantum bits. Shor's algorithm needs a large number of quantum bits in order to run properly and we just don't have enough.

It’s interesting to note that there has been proven that there is a function that is one way <=> one way functions exist. Here’s the paper: https://arxiv.org/pdf/cs/0012023.

PRESENT QUESTIONS

Will public-key cryptography schemes disappear?
I think that the answer is yes, they will disappear. Humans are always striving for progress and, for some scientists, progress means inventing quantum computers advanced enough to sustain Shor's algorithm which, when used, will be able to break keys based on RSA Schemes. We will have to move on from such systems sooner, rather than later.

Do real one-way functions exist?

We have discovered that functions we thought were one-way are actually vulnerable when exposed to a quantum computer, so I believe that we have to reconsider what we categorize as one-way from now on, since science is continuously evolving.

However, the existence of this type of functions is tightly linked with the P vs NP Problem. If we proved that one-way functions do indeed exist, that will mean that there is a problem which is hard to solve, but easy to verify. It would mean that P does not equal NP.

If we could prove that P=NP, that would mean that every problem is solvable, so one-way functions would not exist.

The converses are not known to be true. P ≠ NP would not directly imply the existence of these functions and the non-existence of one-way functions would not directly imply P=NP.

As for our personal opinions, ...

Bibliography

https://en.wikipedia.org/wiki/One-way_function

https://en.wikipedia.org/wiki/RSA_cryptosystem

https://kaustubhrakhade.medium.com/shors-factoring-algorithm-94a0796a13b1

https://datatracker.ietf.org/doc/html/rfc3526#section-3

https://www.youtube.com/watch?v=SL7J8hPKEWY

https://www.proofpoint.com/uk/threat-reference/brute-force-attack

https://freemanlaw.com/hash-collisions-explained/

https://corporatefinanceinstitute.com/resources/cryptocurrency/hash-function/
https://en.wikipedia.org/wiki/Peter_Shor
https://specopssoft.com/blog/sha256-hashing-password-cracking/
https://en.wikipedia.org/wiki/Random_self-reducibility
https://www.classiq.io/insights/shors-algorithm-explained
https://www.cs.princeton.edu/courses/archive/spring08/cos598D/scribe4.pdf
https://arxiv.org/abs/cs/0012023
https://www.cs.cornell.edu/courses/cs6830/2009fa/scribes/lecture21.pdf
https://link.springer.com/article/10.1007/BF00190757
https://cryptography.fandom.com/wiki/One-way_function