Public Key Encryption

COMPUTER NETWORKS A.A. 24/25

Leonardo Maccari, DAIS: Ca' Foscari University of Venice,
leonardo.maccari@unive.it

Venice, fall 2024

Sect. 1 Public Key Encryption: Introduction

Public Key Cryptography

- We have analyzed symmetric key encryption, and we have understood its principles:
  - Alice and Bob share a secret key using some secure channel.
  - The same key is used both for encryption and decryption (that's why it is called symmetric)
- Symmetric key encryption requires a secure channel that guarantees secrecy and authentication, to exchange the key
- then it can be used to secure an insecure channel.

Public Key Cryptography Principles

- With public key cryptography Alice and Bob can communicate securely without exchanging any secret information
- There are many algorithms available, among which the most used ones are:
  - Diffie-Hellman Key exchange
  - RSA public key encryption
  - Elliptic curve encryption (we will not describe this)

Reminder of Discrete Mathematics

- For the sake of the course, it is necessary to recall this notation.
- Consider a and n to be two integer numbers

$$a \mod n = \texttt{a \% n}$$

  that is, the rest of the integer division between a and n.
- Example:

$$1000 \mod 10 = 0$$

$$1000 \mod 11 = 1011 \mod 11 = 10$$

$$123213 \mod 8371 = 6019$$

- Note:
  - n is generally chosen as a prime number
  - $a \mod n < n \; \forall a, n$

Intractability of some Problems

There is no known polynomial algorithm to factor big numbers. This means that if $m = p \times q$ where p and q are primes, and m is made of l digits, there is no known algorithm that can retrieve p and q in at most l^k operations, for some k.

- This is a problem we have been trying to solve for centuries, so there is *some confidence* that such an algorithm does not exist.

Intractability of some Problems

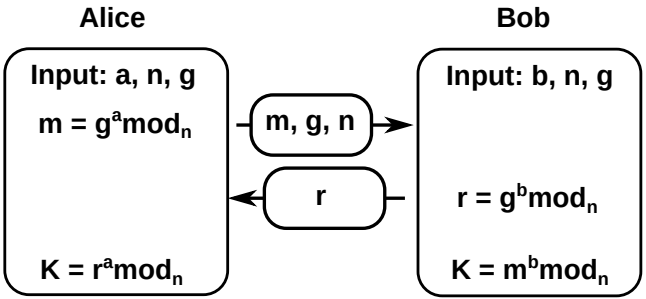
There is no known polynomial algorithm to find the discrete logarithm of a number. It means that given g, n and $g^a \mod n$ there is no known polynomial algorithm to invert the exponentiation and find a.

Sect. 2 Diffie-Hellman

Diffie-Hellman Protocol

- DH is a protocol that allows Alice and Bob to negotiate a symmetric secret key without exchanging any secret information.
- It is based on the difficulty of computing the discrete logarithm.

Diffie-Hellman



- Alice picks n prime, a random, g is a prime root modulo n.
- Bob generates a random value b
- a and b are secret numbers, they never leave Alice and Bob
- Alice sends m, g, n to Bob
- Bob sends r to Alice
- the generated key is $K = g^{ab} \bmod p$

Demonstration

- It is trivial to show that both parties generate the same key.
- We call $K' = r^a \mod n$ and $K'' = m^b \mod n$ we want to show $K' = K''$

$$K' = (g^b \mod n)^a \mod n = (g^{ba} \mod n) \mod n$$

$$= (g^a \mod n)^b \mod n = m^b \mod n = K''$$

- If Eve is able to passively intercept the traffic, she knows m, g, n and r but due to the hardness of the discrete logarithm problem she can not obtain a, nor b.

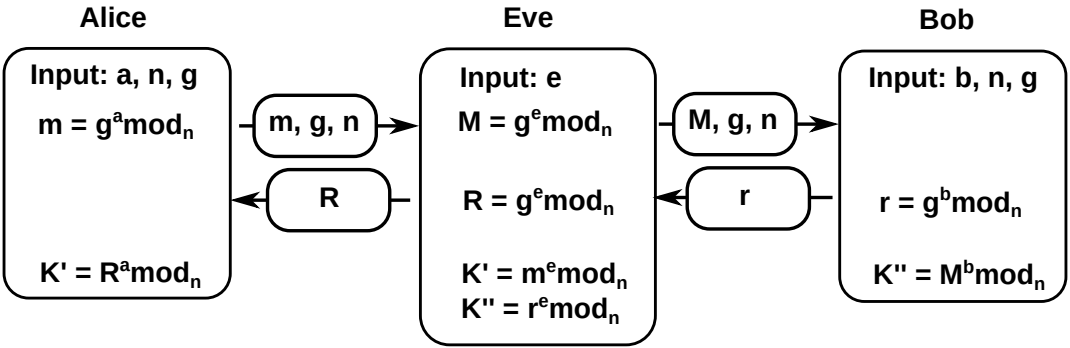
Problem Solved?

- K is a secret piece of data that can be used by Alice and Bob as a key for a symmetric key algorithm
- K was independently generated by Alice and Bob, it was never transmitted on the channel
- In order to generate K, Alice and Bob did not need a previously shared secret, like a password.
- . . . then, problem solved?

Question

- What if, Eve can also modify the traffic, not only passively *sniff* it?

Active MiTM Attack



Active MiTM Attack

- At the end of the exchange, Alice believes she shares a key K' with Bob, while instead she shares it with Eve
- At the end of the exchange, Bob believes she shares a key K'' with Bob, while instead she shares it with Eve

Eve can now read and modify the traffic between Alice and Bob, so DH can guarantee *confidentiality* only if it is used with some other instrument that guarantees *authentication*.

Sect. 3 RSA

Public Key Encryption with RSA

The principles of Public Key Encryption are:

- Alice and and Bob have **two** keys each.
- Alice owns a public key `Pub_A` and a private key `Priv_A`
- Bob owns a public key `Pub_B` and private key `Priv_B`

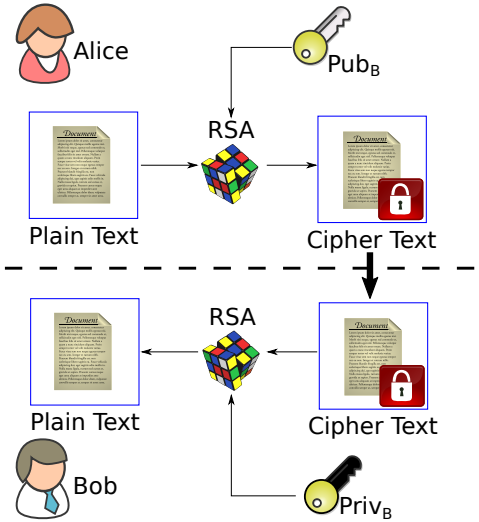
The private key must be kept secret. It is vital that Alice is the only owner of `Priv_A` and the same applies to Bob with `Priv_B`.

The public key instead is public, Alice and Bob can publish `Pub_A` and `Pub_B` anywhere on the Internet

Public Key Encryption

- What is encrypted with a public key can only be decrypted with the corresponding private key
- If Alice uses `Pub_B` to encrypt a message, then only Bob can decrypt it, because Bob is the only one that has the corresponding private key.
- This means literally that if Alice does:
  1. encrypt a message M with `Pub_B`, and obtains an encrypted message C
  2. erase M

  she can not recover M from C, even if Alice encrypted it in first place.


Alice
Pub_B
RSA
Plain Text
Cipher Text
RSA
Plain Text
Cipher Text
Bob
$Priv_B$

Advantages of Public Key Encryption

- Public key encryption provides secrecy, as symmetric key encryption
- The big advantage over symmetric key algorithms is that there is no need for a secret channel to exchange the key.
- All Alice needs to encrypt something to Bob is `Pub_B`, which is public by definition.
- The most known algorithm for public key encryption is called RSA (from the names of the authors).

Is this Possible?

- In practice we are looking for a function $f : D \to R$ with a t parameter that has three features:
  1. f is easy to compute in the direction D to R
  2. f is easy to compute in the direction R to D if you know t
  3. it is theoretically impossible to compute f in the direction R to D if you don't know t
- In this scheme, f would be the public key, and t the private key.
- Unfortunately in math, such a function does not exist.
- In general, mathematical functions are invertible, or they are not. No function exists that we know that is theoretically invertible only if you know some parameter t.
- Again, we have to rely on an approximation of such a function that makes things "computationally impossible"

RSA

The most well known approach to public key crypto is due to Rivest, Shamir and Adelmann, and thus referenced to as RSA. The (simplified) mathematical steps are:

Creation of Keys

- Choose two large prime numbers p and q, they must never be revealed.
- Compute $n = pq$ and Euler's Totient Function: $\phi(n) = (p-1)(q-1)$.
- Choose an integer e such that $2 < e < \phi(n)$ that has no common divisor with $\phi(n)$ (they are coprime).
- Determine d so that

$$de \bmod_{\phi(n)} = 1$$

that is, d and e are inverse in modulo $\phi(n)$.

Public and Private Keys

- (e, n) is the public key. These are two numbers that Alice can publish anywhere.
- d is the private key. This is a number that Alice must keep secret.

RSA Operations

Note Well

- Alice generates `Pub_A` and `Priv_A` at the same moment.
- They are the result of one single mathematical process, there is a mathematical connection between them
- They are not just two random numbers like two symmetric keys, they can not be generated from some known information (like with a hashed password).
- You can not generate them independently one from the other (i.e. in two different places or moments).

Encryption and Decryption

- If Bob wants to send a message M to Alice he will encrypt it as follows:

$$C = M^e \bmod_n$$

- Alice will receive the message and decrypt it as follows:

$$M = C^d \bmod_n$$

Example

- Let's choose $p = 7, q = 11$, then $n = 77, \phi(n) = 60$
- We choose $e = 13$, that is prime and has no common divisor with 60
- We need to find d so that $(d * 13)\text{mod}_{60} = 1$. In real life you would use an efficient algorithm, with small numbers you can make a simple for loop.
- Turns out $d = 37$
- Then, assume $M = 00010100$
- we convert it to integer for clarity, $M = 20$ then you have:

$$C = 20^{13}\text{mod}_{77} = 69 \quad \leftarrow \text{Encryption}$$

$$M = 69^{37}\text{mod}_{77} = 20 \quad \leftarrow \text{Decryption}$$

Public Key Encryption Desired Properties

Let $E()$, $D()$ be the encryption and decryption function: $C = E(\texttt{Pub\_A}, M)$ means C is a ciphertext produced encrypting M with `Pub_A`. We want that:

1. It is computationally efficient to generate `Pub_A`, `Priv_A`.
2. Given `Pub_A` it is computationally efficient to compute $C = E(Pub_A, M)$.
3. Given `Priv_A` it is computationally efficient to compute $M = D(Priv_A, C)$.

Application to RSA

RSA was designed so that there are polynomial algorithms to generate the keys and to encrypt and decrypt a text.

Public Key Encryption Desired Properties (II)

4. Given only `Pub_A` it is computationally impossible to derive `Priv_A`
5. Given `Pub_A` and C it is computationally impossible to derive M (even for Bob that encrypted it in first place).

Application to RSA

- If Eve wants to break RSA given e, n, she should find d
- That means deriving p, q, then computing $\phi(n)$ and inverting $de \mod \phi(n)$
- However, due to the hardeness of integer factoring, this is considered computationally impossible for long enough keys (longer than 1024 bit)

RSA is Complex

- Besides the simplicity of the concept, using RSA in a secure manner is extremely hard
- There are many constraints that need to be respected (random number generators, time padding, etc.)
- So the use of well known, standard and open source libraries is mandatory.
- The crypto geeks that want to go deeper can read `https://crypto.stanford.edu/~dabo/papers/RSA-survey.pdf`

Internet Example

- When Alice signs-up to a new Internet service provided by Bob, she needs credentials (a username and a password)
- Alice can use RSA:
  1. Alice initially downloads Bob's public key and sends to Bob her public key (this is needed only once).
  2. Now Alice and Bob have a secret channel they can use, and Bob can send Alice her credentials (log-in and password).
- Apparently RSA solves all the problems.

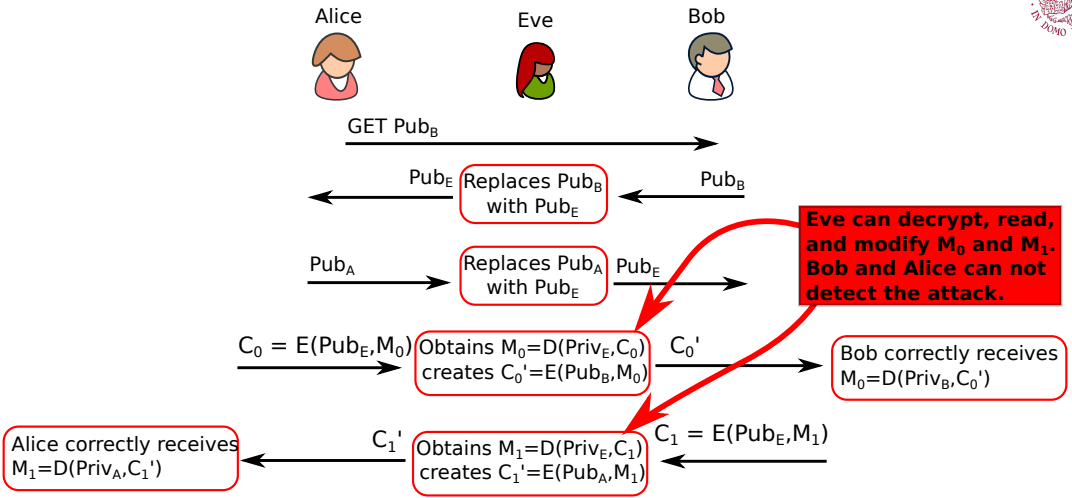
RSA

↳3.1 Limits of Public Key Encryption

Attacks on RSA

Suppose Alice and Bob want to exchange two messages M_0 and M_1. The first thing Alice and Bob need to do is to exchange public keys. Now consider the case in which Eve controls the channel. This can happen:

1. Alice asks Bob for `Pub_B`
2. Eve intercepts the message, exchanges `Pub_B` with `Pub_E`
3. Alice receives the message and is convinced that she received Bob's public key
4. Now when Alice wants to encrypt something for Bob, she will use Eve's key.
5. Eve will decrypt it, encrypt it with `Pub_B` and send it to Bob.
6. Eve can read and modify the message, encrypt it with `Pub_B` and send it to Bob. Bob can not detect the successful attack.
7. Same things happens in the opposite direction.



The attack produces total loss of confidentiality and integrity

Public Key Criptography Limitations

- So Public Key encryption is secure as long as there is a secure channel to exchange public keys.
- If Alice and Bob have it, then MiTM attack is impossible.
- Are we back to the paradox "*to secure a communication channel we need a secure channel*"? **No**.

Public Key Encryption Needs an Authenticated Channel

The major difference between Public key cryptography and Symmetric key cryptography is that the latter needs an authenticated, integer and secret channel, while the former needs **only** an authenticated and integer channel.

Alice and Bob need to exchange keys in a way they are sure that the keys they receive come from one another, but keys are public, not secret.

RSA

↳3.2 Digital Signature

Inverting the keys

RSA has another property that proves to be essential on the Internet:

1. Encryption and decryption can be swapped:

$$\text{if } C = E(\texttt{Priv\_A}, M) \text{ then } M = D(\texttt{Pub\_A}, C)$$

2. What is encrypted with the public key can be decrypted with (and only with) the corresponding private key (we know this already)
3. What is encrypted with a private key can be decrypted with (and only with) the corresponding public key

RSA math

- When using RSA, the digital signature is easy to prove that given that:

$$(M^e)^d \bmod n = M$$

$$\text{if } S = M^d \bmod n \text{ then } M = S^e \bmod n$$

Let's plug the first eq. into the second:

$$(S^e) \bmod n = (M^d)^e \bmod n = (M^e)^d \bmod n = M$$

- So if you encrypt M with d instead that with e you obtain a signature S, if you then decrypt S it with e instead that with d, you obtain M again.

Digital Signature

- If Alice is the sole owner of `Priv_A` and encrypts a message $C = E(\texttt{Priv\_A}, M)$, then someone that owns `Pub_A` can decrypt it

Digital Signature

- If Alice is the sole owner of `Priv_A` and encrypts a message $C = E(\texttt{Priv\_A}, M)$, then someone that owns `Pub_A` can decrypt it
- but `Pub_A` is public, so potentially everyone owns it and thus C can be decrypted by everyone. So why doing it?

Digital Signature

- If Alice is the sole owner of `Priv_A` and encrypts a message $C = E(\texttt{Priv\_A}, M)$, then someone that owns `Pub_A` can decrypt it
- but `Pub_A` is public, so potentially everyone owns it and thus C can be decrypted by everyone. So why doing it?
- Since only Alice is in possession of `Priv_A`, whoever deciphers the message is sure that the message comes from Alice.

Digital Signature

- If Alice is the sole owner of `Priv_A` and encrypts a message $C = E(\texttt{Priv\_A}, M)$, then someone that owns `Pub_A` can decrypt it
- but `Pub_A` is public, so potentially everyone owns it and thus C can be decrypted by everyone. So why doing it?
- Since only Alice is in possession of `Priv_A`, whoever deciphers the message is sure that the message comes from Alice.

Digital Signature

- When used with inverted keys, public and private key encryption is not a guarantee of secrecy but provides authentication.
- This use of public key cryptography is called **digital signature** and provides authentication of data (normally used for origin authentication).

Non Repudiation

- When something is digitally signed, it is non-repudiable.
- Bob received a message that only Alice could generate, because it is signed with her private key.
- Alice can not deny to be the sender of the message.

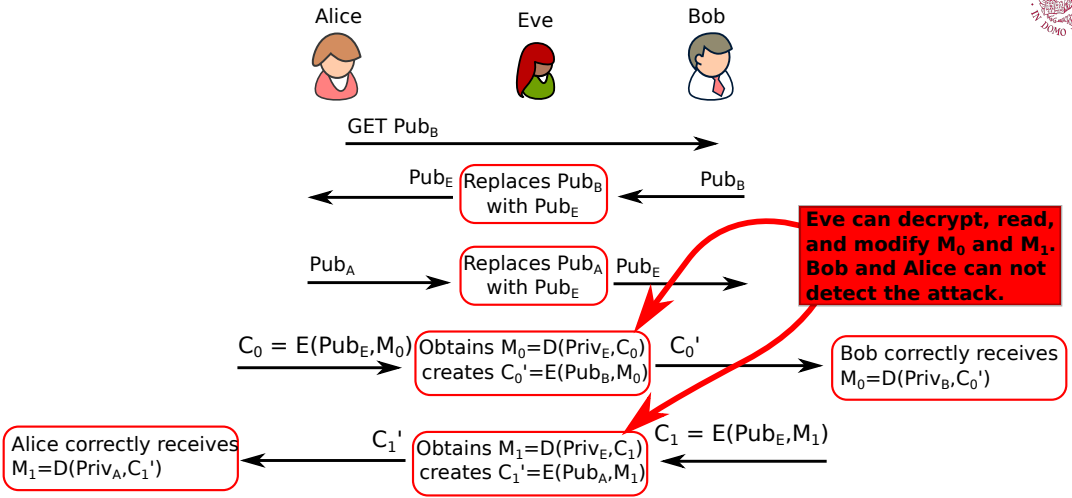
We finally have a security mechanism that can be used to enforce the non-repudiation security service.

Practical application

- In Italy, la Posta Elettronica Certificata works like that
- When you send a PEC you receive a digital signature by the receiving mail server that certifies that you have sent that message.
- Similarly, the receiver has a signature that certifies that he received the message from one verified PEC account.
- This second thing is generally not useful, as PEC is used as a guarantee for the sender.

Legal Value

- In many states (Italy included) there are digital signature providers.
- They provide you with a valid private and public key.
- If you sign a document with that key, that document is legally binding.



Does digital signature eliminate MiTM?

MiTM is still valid

- Note that MiTM can not be defeated using digital signatures
- Before verifying a digital signature, Alice needs to receive Bob's key.
- The MiTM attack takes place at key exchange, thus it is effective against both encryption and authentication.

RSA Properties: summary

- RSA provides secrecy
- RSA provides authentication
- RSA privedes non-repudiation

All this, without the need of a secret channel to exchange a key

Let me Stress this Concept

Shared Key Encryption

Provides:

- Secrecy, Integrity, Authentication

Needs:

- A secret and authenticated (and thus integer) channel to set-up the communication

Public Key Encryption

Provides:

- Secrecy, Integrity, Authentication, **Non-repudiation**

Needs:

- An authenticated (and thus integer) channel to set-up the communication

Focus Shift

- When using symmetric key encryption the problem is “how can Alice give to Bob a secret information”?
- While with Public key, the focus shifts to “how can Alice be sure of the Identity of Bob when she receives `Pub_B`”?
- If Alice and Bob solve this issue, then they can set-up their secure communication channels to distribute public keys.
- This is the topic for the next class

RSA

↳3.3 RSA Performance

Cryptography Speed Test: `openssl speed`

- Signature performs encryption, verify performs decryption
- encryption/verification is faster than decryption/signature.
- Key size for gray rows are too small to be secure
- Signature and Verify operations are done on hashes of size 36 bytes

key bits	sign	verify	sign/s	verify/s
512	0.000048s	0.000003s	20740	333503
1024	0.000103s	0.000007s	9716	149899
2048	0.000708s	0.000021s	1412	47911
3072	0.002114s	0.000043s	473	23413
4096	0.004975s	0.000072s	201	13873

Algorithm	B/s processed
AES-256 cbc	87745k
AES-128 cbc	121855k
sha256	146729k

RSA 2048 encryption achieves $1412 \times 36 \times 8 \simeq 0.4Mb/s$ against $\simeq 87 \times 8 \simeq 696Mb/s$ of AES256: three orders of magnitude slower.

Reasons for Asymmetric Performance

- Recall that $de \mod_{\phi(n)} = 1$, so $de = k * \phi(n) + 1$ for some k, and that $\phi(n)$ must be a big number.
- This means that if e is *small*, then d needs to be *big* (or the other way around)
- d **must** be big, because it must be secret and hard to guess, so e could be small

Asymmetric Performance

- Since you are free to choose e, most of the implementations use a fixed, small value $e = 2^{16} + 1 = 65,537$ that is prime.
- Since encryption and decryption is an exponentiation (the same math operation) if you use a small exponent, it takes less time than if you use a big one.
- So the idea of setting $e = 2^{16} + 1 = 65,537$ makes encryption faster than decryption.

Limits of RSA: low performance

- Plus, since decryption is harder than encryption, using RSA to encrypt traffic helps an attacker that wants to make a DoS attack.
- because, given a certain traffic exchanged, the server that decrypts the traffic needs more resources than a client that encrypts it.
- in general public key encryption is *fast* enough to be usable but not as fast as symmetric key encryption. Encrypting large files takes ages.

Practical Consequences

Mixed Mode

You generally avoid to encrypt/decrypt data with RSA. What is normally done is to use RSA to communicate a secret key, and then the secret key is used with symmetric key encryption and HMAC.

Signature of Hashes

Again, for performance reason, when Alice wants to digitally sign a message M, she generally does not encrypt the whole M with `Priv_A`, She produces a digest T of M and encrypts only T:

- $C = E(\texttt{Priv\_A}, T = H(M))$.
- When Bob receives both M and C, he will compute $T' = D(\texttt{Pub\_A}, C)$ and check if $T' = H(M)$

Example (dummy) Protocol

1. A $\rightarrow$ B: key of A
2. A $\leftarrow$ B: key of B
3. A $\rightarrow$ B: A generates a random number R, transmits it encrypted with `Pub_B`
4. A symmetric key is generated by both with a hash function: $K = hash(R)$
5. From now on Alice and Bob stop using the public keys and use AES and HMAC with K
6. Note: this is just an example, not a real protocol, it suffers of MiTM. TLS does something similar, but way more complex (material for future lessons).

RSA

↳3.4 A few Notes on PK security

RSA Vs AES: Fundamental Difference

- Symmetric key encryption is some kind of approximation of a One-Time-Pad.
- We know that OTP is theoretically secure, and AES mimics its behaviour in a consistently robust way. We know the math is simple.
- RSA is based instead on a hard mathematical problem.
- "Hard" is parametrized on to two things:
  - The best algorithm we know that solves the mathematical problem
  - The computational model we use today (i.e. how we computers work).
- With time passing, we produced better algorithms and faster computers and to keep up with the security level we needed to enlarge the keys.

Future Risks

1. Someone finds a new algorithm for integer factorization, and the math falls apart alltogether.
2. Someone accumulates an unthinkable computer power that can break keys that we believe are safe
3. Someone changes the way we make computers and there is a quantum leap in the processing power

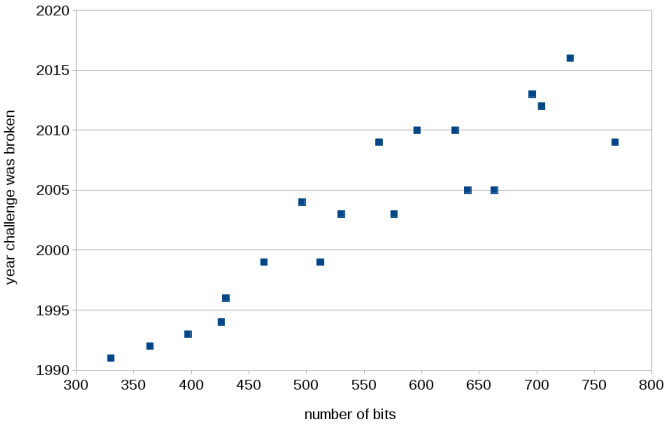
Let's analyse them one by one.

New Algorithms

- We have been trying to factor integers for at least the last 400 years.
- There is a certain consensus on the fact that integer factorization is intrinsically hard, so that any algorithm can improve the state of the art only with a small speed-up
- It seems extremely unlikely that a new algorithm will suddenly make the task *easy*

RSA key breaking challenge

- The capacity of breaking RSA keys increased with years passing
- Keys need to grow as a consequence



More Computational Power

- In the files published by Edqard Snowden, there was a sentence that referred to a decryption program financed with 800M$, which said:
  "vast amounts of encrypted Internet data which have up till now been discarded are now exploitable"
- This sentence was interpreted as the fact that NSA was able to break RSA keys up to 1024 bit.
- Of course there is no official confirmation, but from that day, 1024 bit RSA keys are considered insecure, and must not be used.
- This may happen again in the future, or may be happening now in some other country we don't know about.

New Computation Models

- In the last 40 years researchers have spent an incredible amount of resources on Quantum Computing.
- A quantum computer uses a fully distinct computational model, replacing bits with qbits and allowing massive parallel execution of fundamental operations
- Once these computer are realized, and they support thousands of qbits, the Shor Algorithm will allow to break RSA keys in linear time with their size.
- Today we have quantum computers with a few qbits, and some researchers are critical on the possibility to have usable quantum computers in the future.