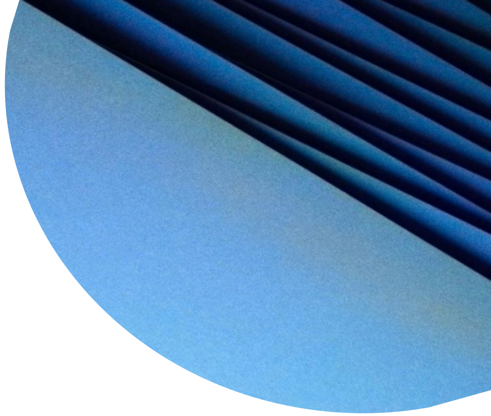
Is RSA Cryptosystem Secure?

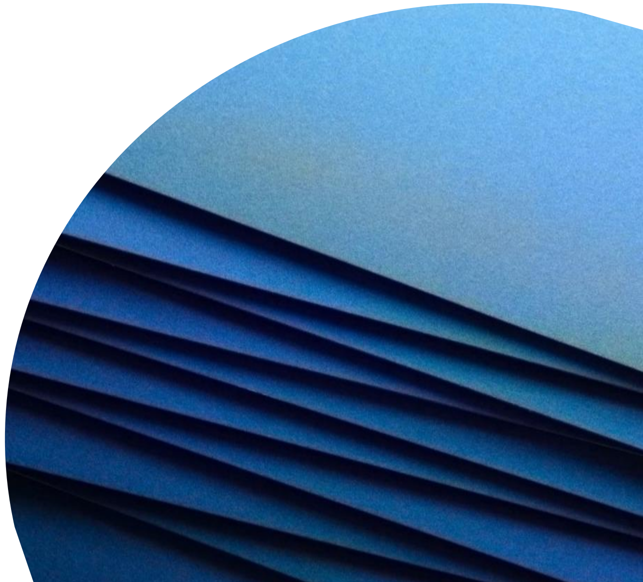
The RSA problem and most famous attacks to RSA

Vaccarezza Gaia 3073561

What is RSA Cryptosystem



- It is an **asymmetric (or public key) cryptosystem.**
- It is also one of the oldest.
- The acronym "RSA" comes from the **surnames** of the inventors.
- Applications:
  - **encrypts messages** sent between two communicating parties so that an eavesdropper who overhears the it will not be able to decode them.
  - enables a party to append an unforgeable **“digital signature”** to the end of an electronic message.
- Examples of use:
  - web servers and browsers to secure web tracking
  - ensure privacy and authenticity of email
  - secure remote login sessions
  - heart of electronic credit-card payment systems and ID documents (“**smart card**”)
  - where security of digital data is a concern.



History

- **Public Key Cryptography**
  - Proposed in Diffie and Hellman in “New Directions in Cryptography” (1976).
  - Public-key encryption was proposed in 1970 by James Ellis
    But Diffie-Hellman key agreement and concept of digital signature are due to Diffie & Hellman.

- **RSA Cryptosystem**
  - Ron Rivest, Adi Shamir and Leonard Adleman publicly described the algorithm in 1977.
  - At first their formulation used a **shared-secret-key** created from exponentiation of some number, modulo a prime number.
  - However, they left open the problem of realizing a one-way function.
    They tried many approaches, including "knapsack-based“ and "permutation polynomials".
  - During Passover, Rivest got drunk and unable to sleep started thinking about their one-way function. He spent the rest of the night formalizing his idea.
    → He come out with the RSA function that is an example of trapdoor one-way function.

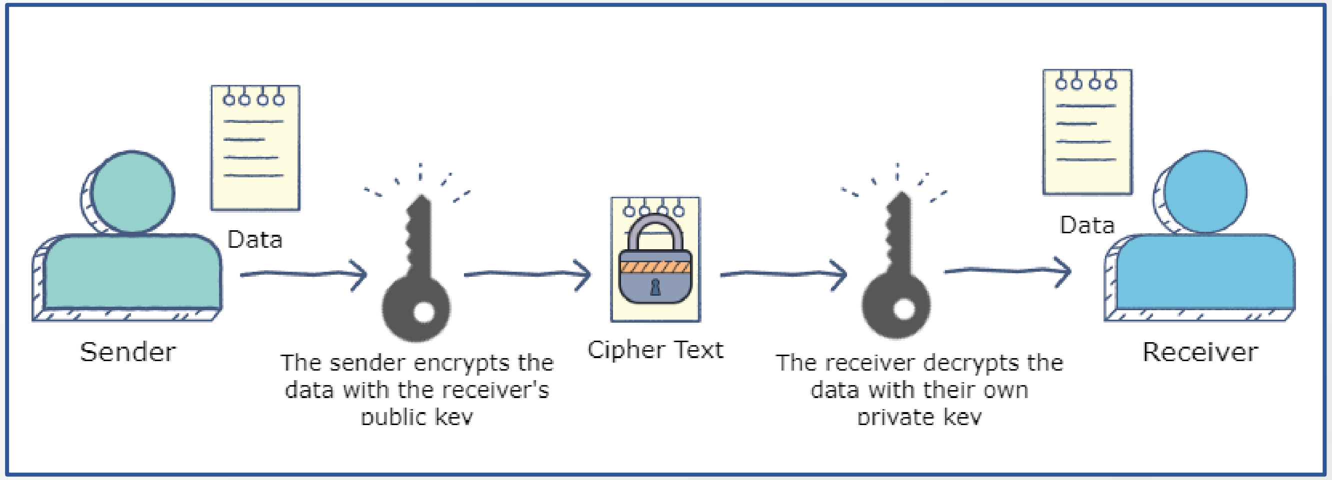
$$x \rightarrow x^e \bmod N$$

  - An equivalent system was developed secretly in 1973 at GCHQ (the British signals intelligence agency) by the English mathematician Clifford Cocks. That system was declassified in 1997

Public and Private Keys

Select	Select **at random** 2 **large prime** numbers p and q with $p \neq q$
Compute	Compute $n = p \cdot q$
Select	Select e as odd integer relatively prime to $\varphi(n) = (p-1) \cdot (q-1)$
Compute	Compute d as the **modular multiplicative inverse** of $e \; mod \; \varphi(n)$ that is $d \equiv e^{-1} \; (mod \; (n))$;
Publish	Publish (e, n) and keep secret $(d, p, q, \varphi(n))$

Security of RSA



Encryption function $\boldsymbol{M \rightarrow M^e \, mod \, N}$

Must be a trapdoor one-way function
that means:

- $f(x)$ is a one-way function

  i.e. knowing $y = f(x)$ it should be difficult to find x.
- However, given some **extra information** it becomes feasible to compute the inverse.

Decryption function $\boldsymbol{C \rightarrow C^d \, mod \, N}$

- Everyone can encrypt but only those having the **secret key** can decrypt in a **reasonable amount of time** (if certain conditions hold).

RSA Problem

The RSA problem is defined as the task to find plaintext from ciphertext.

Exhaustive search?
It is too computational costly under certain condition.
Why? this results in an algorithm with a running time **exponential** in the size of its input.

Other algorithms?
*As today the most promising approach to break it is to use **integer factorization**.*

Why Factorization?
An attacker factors n into p and q, computes $\varphi(n) = (p-1) \cdot (q-1)$ that allows the determination of d from e, and then decrypt c using the standard procedure.

However, integer factorization (under certain conditions) is also **"hard".**

These schemes are

- **feasible**
  because we can find large primes easily.
- **secure**
  because we do not know how to factor the product of large primes (or solve related problems, such as computing discrete logarithms) efficiently.

Integer Factorization

- **Integer factorization** is the decomposition of a composite number into a product of smaller integers.
- Not all numbers of a given length are equally hard to factor.
  The **hardest** instances are **semiprimes**, the product of two prime numbers.
  Especially when the factors are both:
  - **large** (more than two thousand bits long),
  - **randomly chosen**,
  - about the **same size**
  - not too close
- No algorithm has been published that can factor integers of this class of numbers in **polynomial time, with a classic computer**.

But..

..the security is based on:

- RSA problem not having a polynomial-time algorithm to solve it (under certain conditions)
BUT **no proof of non-existence**
YET **no proof of existence**
- RSA problem being at least **as hard as** factoring
BUT **no proof** (known by inventors)
YET **no proof that is easier**
- Based on Integer factorization having no polynomial algorithm
BUT **no proof of non-existence**
YET **no proof of existence**

Integer Factorization Problem

- *Suspected* to be not P
- The problem is in **class NP.**
- Believe not **NP-hard.**
- *Suspected* that it is **not NP-complete** but also not known to be **NP-complete.**
- It is known to be in **BQP** because of Shor's algorithm.

WOULD FINDING A POLYNOMIAL TIME ALGORITHM STOP ALL ASYMMETRIC CRYPTOSYSTEMS IN THE WORLD?

First, a polynomial-time algorithm could still be **too slow** to be practical because the **degree of the polynomial** were high.

Second, **discrete-logarithm-based cryptography** does not depend on the difficulty of integer factorization. That includes Diffie-Hellman, ElGamal, DSA, SRP, and elliptic-curve methods.

Does proving $P = NP$ break cryptography?

- Fundamentally, **the P versus NP problem asks "can every problem whose solution can be "easily" verified also be quickly solved?"** Many systems in cryptography are secure only if the answer to that question is "No". This will break many current cryptographic algorithms**, but overall does not break cryptography as a concept.**
- Public-key cryptography is secure because an attacker must perform a "hard" task in order to discover the private key needed to decrypt the message from the public key. If there was proof that $P = NP$, then this proof would essentially say that there exists some algorithm for doing this "quickly" and efficiently.
- **However,** it would not give one!

Attacks Based On Factoring

Brute force attack

WHAT?
searching for p, and q by trying all possibilities.

HOW?
The size of the set of possible factors can be decreased by
-finding the square root of n
-excluding even numbers and numbers ending in 5

SOLUTION?
pick large primes p and q

«I hope RSA applications would have moved away from 1024-bit security years ago, but for those who haven't yet: wake up.»

(Bruce Schneier, sbsuneebxu21 maggio 2007)

Attacks Based On Factoring

Special-purpose Factoring Methods

WHAT?
Attacks that factoring n using special-purpose factoring algorithms
These are more efficient than general purpose ones if p and q are in the right format.
But generally work for **low numbers**

HOW?
- Pollard's $p-1$ method
- Elliptic curve method (50-60 digits factors)
- Trial division (small factors)
- Fermat's factorization method

Their running time of depend sever properties of number or factor

SOLUTION?
A simple defense against this algorithm is to make n large and the factors the same large size, since the algorithm starts with small factors first
→Generally used before general-purpose algorithm to remove small factors

```python
def trial_division(n: int) -> List[int]:
    a = []
    while n % 2 == 0:
        a.append(2)
        n //= 2
    f = 3
    while f * f <= n:
        if n % f == 0:
            a.append(f)
            n //= f
        else:
            f += 2
    if n != 1: a.append(n)
    # Only odd number is possible
    return a
```

Attacks based on factoring

General purpose factoring methods

WHAT?
attacks performed using general purpose algorithm can be used to factor **any numbers.**

HOW?
- Sieve of Eratosthenes
- Dixon's algorithm
- Continued Fraction factorization
- Quadratic sieve method
- General number field sieve

Running time depends only on size of integer

Most of them are on based on **congruence of squares.**

$$x^2 - y^2 = n$$

$$x^2 \equiv y^2 \pmod{n}$$
$$x \not\equiv \pm y \pmod{n}$$

$$x^2 - y^2 \equiv 0 \pmod{n}$$
$$(x+y)(x-y) \equiv 0 \pmod{n}$$

The General Number Field Sieve

- **Best** theoretical asymptotic running time algorithm that can factor integers larger than 100 digit.
- The largest such semiprime yet factored was RSA-250, a **829-bit** number with 250 decimal digits, in February 2020.
  The total computation time was roughly **2700 core-years** of computing using Intel Xeon Gold 6130 at 2.1 GHz.
- It is **faster than sub-exponential** but still not polynomial.
- An **improvement** to the simpler rational sieve or quadratic sieve
  WHY? manages to search for smooth numbers that are subexponential in the size of n.
- *SOLUTION?*
  use RSA numbers of **1024 to 4096 bits**

$$O\left(e^{1.9(\log N)^{1/3}(\log\log N)^{2/3}}\right)$$

Attacks based on factoring

Factoring on a Quantum Computer

WHAT?
Attacks based on algorithm using quantum computer.

HOW?
Using Shor's algorithm.

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

SOLUTION?
Because the period-finding subroutine must be tuned to each unique value of N and generally speaking, quantum computing is still much more an area of research than a scalable, deployable technology there is **no need to be worried.**

Shor’s Algorithm

In 1994, Peter Shor showed that a **quantum computer** would be able to factor in polynomial time breaking RSA

In 2001, Shor's algorithm was **implemented** for the first time, by using NMR techniques on molecules who factored 15

The efficiency of Shor's algorithm is due to the efficiency of the **quantum Fourier transform**, and **modular exponentiation by repeated squaring**

The algorithm consists of an **iterative process** of generating a random number a and computing gcd(a,N)

Like all quantum computer algorithms, Shor's algorithm is **probabilistic:** it gives the correct answer with high probability, and the probability of failure can be decreased by repeating the algorithm

Attack - Secret key d

WHAT?

Attack that exploit the **low value of $\boldsymbol{d}$** and get it through public key.

HOW?

To reduce decryption time (or signature-generation time), one may wish to use a small value of d rather than a random d.

WHY? A small d can improve performance by at least a factor of 10 (for a 1024 bit modulus).

BUT

Theorem 2 (M. Wiener) *Let $N = pq$ with $q < p < 2q$. Let $d < \frac{1}{3}N^{1/4}$. Given $\langle N, e\rangle$ with $ed = 1 \bmod \varphi(N)$, Marvin can efficiently recover d.*

- Based on **continued fraction approximation**, using the public key (n, e) to provide sufficient information to recover the private key d.

SOLUTION?:

→ d have to be **larger than** $⅓ \cdot N^{0.25}$ (At least 256 bits long)

Attack – Secret key d

BUT this solution is unfortunate for **low-power devices** such as “smartcards", where a small d would result in big savings.

SOLUTIONS?

- A simple calculation shows that **if** $\boldsymbol{e' > N^{1.5}}$ then no matter how small d is, the above attack cannot be done.
  BUT large values of e result in increased encryption time.

OR

- Chooses d such that both $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$ are small, say 128 bits each but d is not small.
  Then fast decryption of a ciphertext C can be carried out as follows:
  - compute $M_p = C^{d_p} \bmod p$ and $M_q = C^{d_q} \bmod q$.

  -Then use the **Chinese Remainder Theorem** to compute the unique value M satisfying $M = M_p \bmod p$ and $M = M_q \bmod q$.
  The resulting M satisfies $M = C^d \bmod N$ as required.

  BUT there exists an attack enabling an adversary To factor N in time $O\left(\min(\sqrt{d_p}, \sqrt{d_q})\right)$.
  So d_q and d_p cannot be too small.

- The theorem was recently improved by Boneh and Durfee, who show that as long as $\boldsymbol{d < N^{0.292}}$
  But they believe that the correct bound is $\boldsymbol{d < N^{0.5}}$.
  This is an **open problem.**

Attacks – Public key e

- The ***Coppersmith method***, is a method to find small integer zeroes of univariate or bivariate polynomials modulo a given integer. The method uses the Lenstra-Lenstra-Lovász lattice basis reduction algorithm (LLL) to find a polynomial that has the same zeroes as the target polynomial but smaller coefficients.

- Theorem

  Let N be an integer and $f \in \mathbb{Z}[x]$ be a monic polynomial of degree d over the integers. Set $X = N^{\frac{1}{d}-\epsilon}$ for $\frac{1}{d} > \epsilon > 0$. Then, given $\langle N, f \rangle$, attacker (Eve) can efficiently find all integers $x_0 < X$ satisfying $f(x_0) \equiv 0 \pmod{N}$. The running time is dominated by the time it takes to run the LLL algorithm on a lattice of dimension O(w) with $w = \min\left\{\frac{1}{\epsilon}, \log_2 N\right\}$.

- *Solutions*:
  Choose $\boldsymbol{e = 2^{16} + 1}$ requires less multiplications, as opposed to roughly 1000 when a random e is used
  and use **long random padding**

Attacks – Public key e

Hastad's Broadcast Attack

- <u>Same message M sent to k people</u>
- $\boldsymbol{e} \leq \boldsymbol{k}$ and equal for all people but different p and q
-Ni differs for each i and gcd(N_i,N_j)=1 for all1,j pairs
-$M < Ni$ for all i

If attacker intercept at least e cyphertexts then he can find a C s.t. $C \equiv Ci\ (mod\ Ni)$
Then $C \equiv M^e\ (mod\ N_1 * \cdots * Ne)$ by the Chinese remainder theorem
and as $M < Ni$ for all i $M^e < N_1 * \cdots * Ne$ we have $C = M^e$
Therefore, we can invert RSA **taking the e-th root**.

SOLUTIONS?
One could pad the message prior to encryption different for each party using **linear padding.**
BUT if the attacker learns (at least e) $C_i = fi(M)^e$ with fi being a linear function.
Then he gets a system of univariate equations modulo relatively prime composites can be efficiently solved. Through these we can find through Chinese remainder theorem g_i() st.
Therefore $g_i(M) \equiv 0$ mod Ni and Coppersmith's method can be used to get M.

→ Use **randomized padding**.

$$g_i = \left(f_i(x)\right)^{e_i} - C_i \bmod N_i\,.$$

Attacks – Public Key e

Franklin-Reiter Related Message Attack

-Linearly related encrypted messages using the same e and N

Then attacker can recover M_1 and M_2 in time quadratic in $\boldsymbol{e} \cdot \log(N)$

$$g_1(x) = f(x)^e - C_1 \in \mathbb{Z}_N[x]$$

WHY?

$$g_2(x) = x^e - C_2 \in \mathbb{Z}_N[x]$$

Since $C_1 \equiv M_1^e$ (mod N) that M_2 is a root of the polynomials:

Then calculating the gcd of g_1 and g_2 we get M_2

Coppersmith's Short Pad Attack

This shows that even random padding may be not enough if e is low

$$h(y) = \mathrm{res}_x(g_1, g_2) \in \mathbb{Z}_N[y]$$

-Two cyphertext of same M but different random padding r

$g_1(x, y) = x^e - C_1$ and $g_2(x, y) = (x + y)^e - C_2$

-$r_1 \geq 0$ and $r_2 < r^m$ $(m = n/e^2)$

-M is $n - m$ $long$ $(n = bits\ of\ N)$

Then can efficently find M using Coppersmith method to find delta $(\Delta = r_2 - r_1)$ $from\ h\ mod\ N$

and then obtain M

Attacks – factors p q

- ***If either p − 1 or q − 1 has only small prime factors***
  n can be factored quickly by **Pollard's p − 1 algorithm** getting d → such values of p or q should be discarded.

- ***If p and q are too close***
  The numbers p and q should not be "too close", to prevent the **Fermat factorization** for n be successful.
  If **(p − q) ≤** $\mathbf{2n^{(0.25)}}$ solving for p and q is trivial

- ***If p and q not enough "randomly generated"***
  -If $n = pq$ is one public key, and $n' = p'q'$ is another, then if by chance $p = p'$ (but q is not equal to q'), then a simple computation of $\gcd(n, n') = p$.

  - Compute the GCD of each RSA key n against the product of all the other keys n' they had found (a 729-million-digit number), thereby achieving a very significant speedup, since after one large division, the GCD problem is of normal size.

  SOLUTIONS?
  Use a **cryptographically strong random number generator**, which has been properly seeded with adequate entropy and by employing a **deterministic function** to choose q given p, instead of choosing p and q independently.

Attacks – same N

Same N for all users

The same N is used by all users and different ei and di for each user. Bob can use his own exponents ei,di to factor the modulus N. Once N is factored he can get Alice d from her public key e.

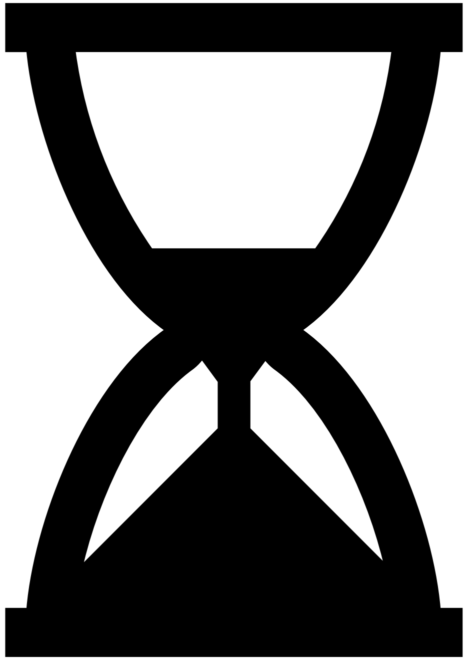
→ RSA modulus N should never be used by more than one entity.

Side Channel Attacks

- **Side-channel attack** is any attack based on information gained from the implementation of a computer system

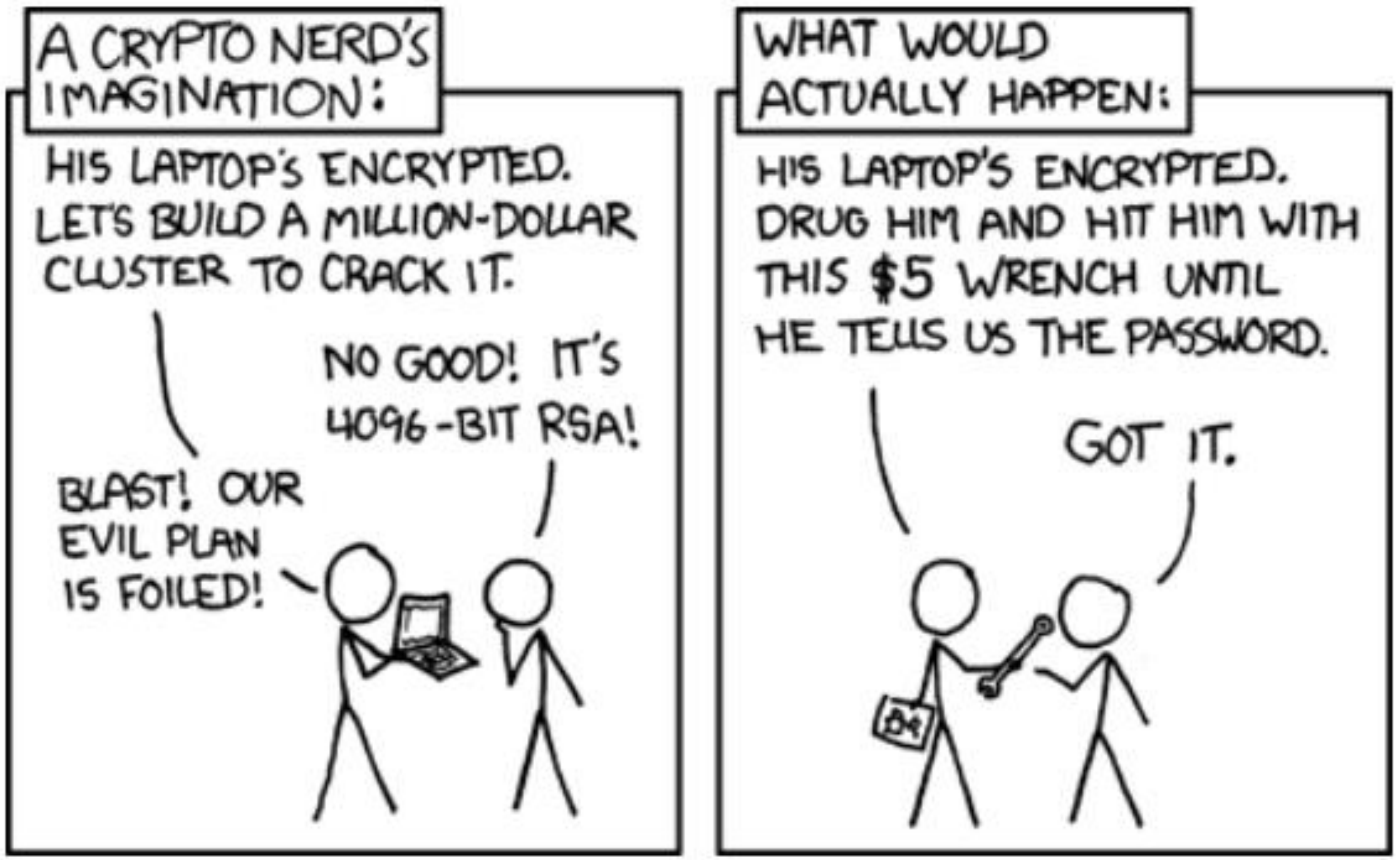
- ***Timing Attacks***
  If the attacker knows the receiver's hardware in sufficient detail and is able to measure the decryption times for several known ciphertexts, he can deduce the decryption key d quickly.
  WHY? The execution time for the square-and-multiply algorithm used in modular exponentiation **depends linearly on the number of '1' bits** in the key. Repeated executions with the same key and different inputs can be used to perform **statistical correlation** analysis of timing information to recover the key completely.

  SOLUTIONS?
  -Add appropriate delay so that modular exponentiation always takes a fixed amount of time.
  -Blinding makes use of the multiplicative property of RSA. Instead of computing $c^d \pmod{n}$, the receiver first chooses a secret random value r and computes $(r^e c)^d \pmod{n}$.
  With blinding applied, the decryption time is no longer correlated to the value of the input ciphertext, and so the timing attack fails.



Side Channel Attacks

- **Power attacks**
  measuring the smartcard's power consumption during signature generation.
  WHY?
  during a multi-precision multiplication the card's power consumption is higher than normal. By measuring the **length of high consumption** periods, Marvin can easily determine if in a given iteration the card performs one or two multiplications, thus exposing the bits of d.
- **Electromagnetic attacks**
  attacks based on leaked electromagnetic radiation.
- **Acoustic attacks**
  attacks that exploit sound produced during a computation.



Thanks for the Attention!

SOMETHING TO NOTE

- “Large input” typically means an input containing “large integers” rather than an input containing “many integers” (as for sorting).
  → we shall measure the size of an input in terms of the number of **bits required to represent that input**, not just the number of integers in the input.
- Elementary operations can be time-consuming when their inputs are large.
  → becomes convenient to measure how many bit operations a number-theoretic algorithm requires.
- An algorithm with integer inputs a1; a2...;ak is a **polynomial-time** algorithm if it runs in time polynomial in the lengths of its binary encoded input.