# Homomorphic Encryption:
## Manipulating Data while it is Encrypted

Craig Gentry
IBM T.J. Watson Research Center

July 10, 2018

SIAM: Portland, Oregon

# "Homomorphic Encryption" at a High Level

A way to delegate processing of your data,
without giving away access to it.

# Other Applications

- Private Google search
  - Encrypt my query, send to Google
  - Google answers my query without seeing it
  - Google's response is also encrypted

# Other Applications

- Private Google search
  - Encrypt my query, send to Google
  - Google answers my query without seeing it
  - Google's response is also encrypted
- Private online tax return preparation
- Encrypted artificial intelligence

# Does Homomorphic Encryption Seem Impossible?

Actually, separating <u>processing</u> from <u>access</u> makes sense even in the physical world…

# An Analogy: Alice's Jewelry Store

- Workers assemble raw materials into jewelry

# An Analogy: Alice's Jewelry Store

- Workers assemble raw materials into jewelry
- But Alice is worried about theft

  How can the workers <u>process</u> the raw materials without having <u>access</u> to them?

# An Analogy: Alice's Jewelry Store

- Alice puts materials in locked glovebox
  - For which only she has the key
- Workers assemble jewelry in the box
- Alice unlocks box to get "results"

# An Encryption Glovebox?

- Alice delegated <u>processing</u> without giving away <u>access</u>.
- But does this work for encryption?
  - Can we create an "encryption glovebox" to securely process data while it remains encrypted?

# The Homomorphism in HE

$\mathcal{M}$ = set of messages, $\mathcal{C}$ = set of ciphertexts

$C_1 = Enc(m_1), \ldots$
$C_t = Enc(m_t)$

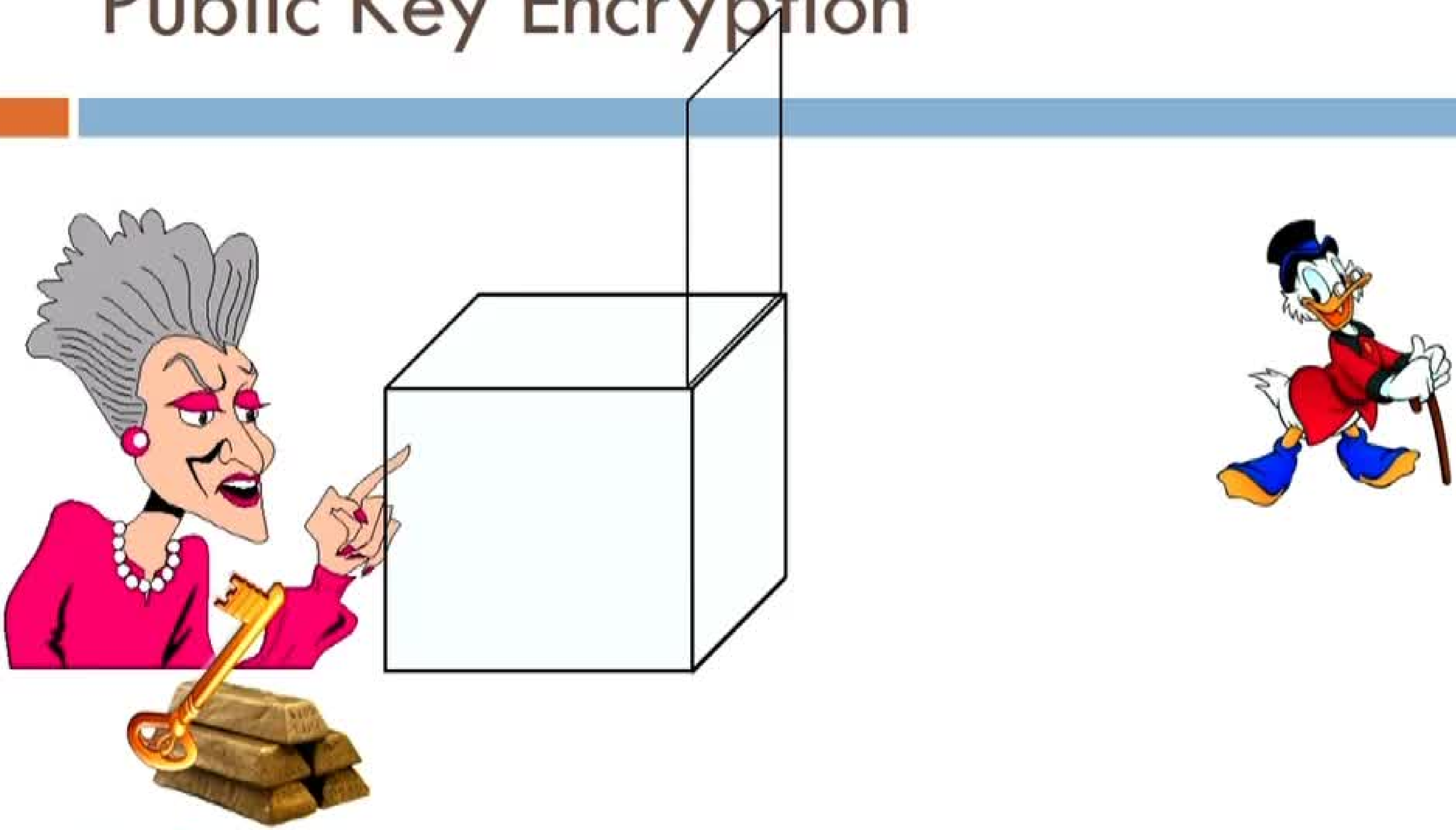
$$\begin{tikzcd}
\mathcal{C}^t \arrow[rr, "Eval(pk, f, \cdot, \ldots, \cdot)"] \arrow[d, "Dec(sk, \cdot, \ldots, \cdot)"'] & & \mathcal{C} \arrow[d, "Dec(sk, \cdot)"] \\
\mathcal{M}^t \arrow[rr, "f(\cdot, \ldots, \cdot)"] & & \mathcal{M}
\end{tikzcd}$$

$f(m_1, \ldots, m_t)$

For any key, messages, ciphertexts, and function $f$, the order of $f$ and Decryption doesn't matter: either way we get $f(m_1, \ldots, m_t)$.

# Public-Key Encryption

# Public Key Encryption



- Key Generation: Alice uses randomness to generate a key pair (pk, sk). She publishes pk and keeps sk secret.
- Encryption: c ← Enc(pk, m) to get a ciphertext c that encrypts message m.
- Decryption: m ← Dec(sk, c) to obtain m.

# Security of Public-Key Encryption

- Semantic security: For any $m_0 \neq m_1$,

  $$(pk, \mathrm{Enc}_{pk}(m_0)) \approx (pk, \mathrm{Enc}_{pk}(m_1))$$

  - $\approx$ means indistinguishable by efficient algorithms.

- Any semantically secure encryption scheme must be probabilistic – i.e., many ciphertexts per plaintext.

- But what does "indistinguishable by efficient algorithms" mean?

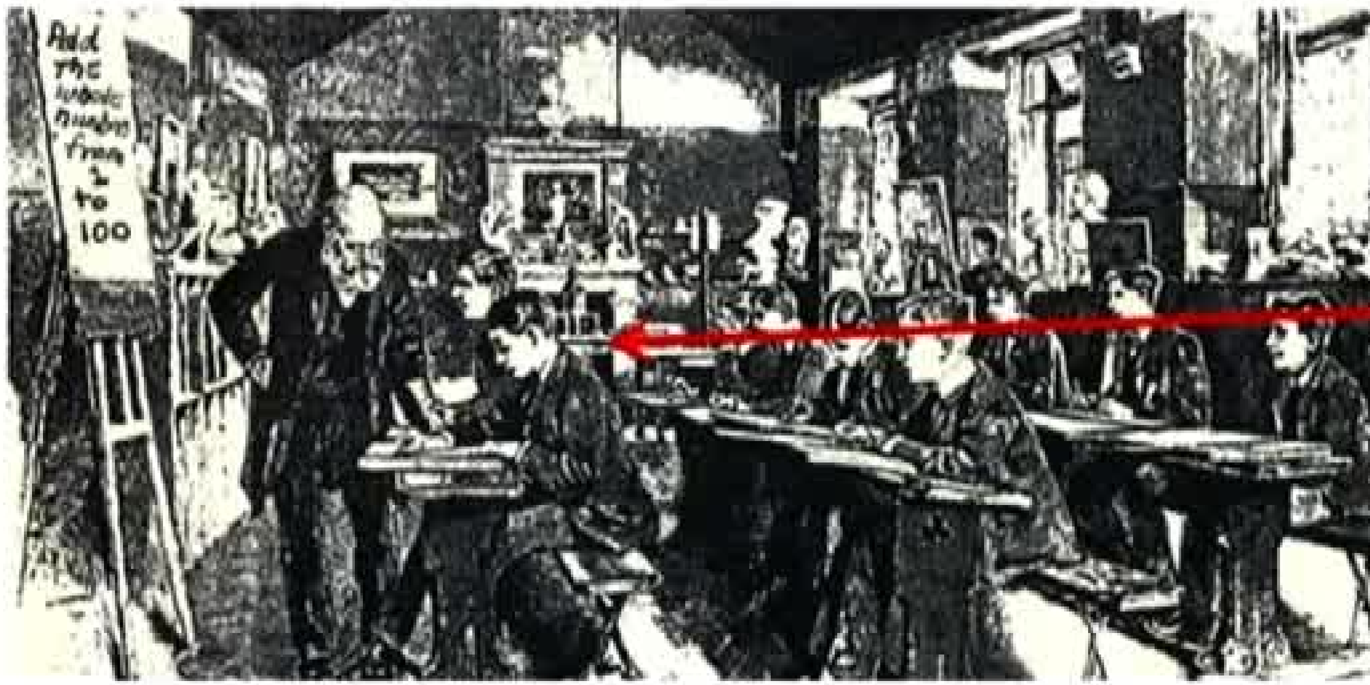
# Algorithms and Computational Hardness

# Are You Smarter than a $5^{th}$ Grader?

What is $1 + 2 + 3 + \ldots + 100$?

# Are You Smarter than a 5th Grader?

What is 1 + 2 + 3 + …+ 100?



Gauss

$$
\begin{array}{r}
 1 \quad 2 \quad 3 \quad 4 \quad \cdots \quad 50 \\
+ \; 100 \quad 99 \quad 98 \quad 97 \quad \cdots \quad 51 \\
\hline
\underbrace{101 \quad 101 \quad 101 \quad 101 \quad\quad 101}_{50} \rightarrow 5050
\end{array}
$$

Algorithm!

$1 + 2 + 3 + \ldots + n = n(n+1)/2$ (formula)

# Algorithms: Efficient vs Inefficient

Efficient algorithm: Takes time $\leq$ polynomial in length of input.

Sum 1 to n ← Length of input n is $k = \log_2 n$ bits (or $\log_{10} n$ decimal digits).

Gauss' algorithm (multiplication) takes $O(k^2)$ steps.

Polynomial in input length

Inefficient algorithm: not polynomial-time.

Other students' algorithm takes about $n = 2^k$ steps.

Exponential in input length

# P vs NP

**P**: Class of problems solvable by poly-time (efficient) algorithms

Examples: sum 1 to n, multiplication of two numbers

**NP**: "Non-deterministic polynomial-time"

Class of problems that, if you guess a solution, you can verify it in polynomial-time (efficiently).

# P vs NP

**P**: Class of problems solvable by poly-time (efficient) algorithms

Examples: sum 1 to n, multiplication of two numbers

**NP**: "Non-deterministic polynomial-time"

Class of problems that, if you guess a solution, you can verify it in polynomial-time (efficiently).

Example: Factoring (factor n into its prime factors)

In **NP**: Given primes p and q, can check n = pq in poly-time.

Not in **P** (we think): No poly-time algorithm to *find* p and q.

**P vs NP Question**: Prove **P** $\neq$ **NP** (if that is the case)

Big open problem in mathematics / CS ($1 million prize)

# Cryptography and P vs NP

Breaking public-key encryption (Is it in **NP**? In **P**?)

In **NP**: Guess randomness r used in key generation. Verify that r generates (pk, sk) where pk is public key. Decrypt with sk.

Hopefully not in **P**: Best breaking algorithms should take exponential time: time $2^\lambda$, where $\lambda$ is a "security parameter".

Ciphertexts "indistinguishable by efficient algorithms"

Secure public-key encryption exists only if **P** $\neq$ **NP**!
Big unproven assumption!

# "Provable Security"

In modern cryptography, we try to *prove* our cryptosystems secure based on a natural, plausible *assumptions*.

Example: For some encryption schemes, we can prove:

1) If there is an efficient algorithm to break it,
2) Then there is an efficient algorithm to factor integers.

What assumptions are plausible and natural?

# Good Assumption for Crypto? Factoring

Factoring: Given k-bit integer n, output a nontrivial factor of n.

Best-known algorithm: The "Number Field Sieve" takes $2^{O(k^{1/3} (\log k)^{1/3})}$ steps (sub-exponential in input length).

Quantum algorithm: Uses principles of quantum mechanics.
Quantum computers can factor in poly-time!! [Shor, 1993].
Quantum computers break most public-key cryptosystems in use!

# Good Assumption for Crypto? Factoring

Factoring: Given k-bit integer n, output a nontrivial factor of n.

Best-known algorithm: The "Number Field Sieve" takes $2^{O(k^{1/3} (\log k)^{1/3})}$ steps (sub-exponential in input length).

Quantum algorithm: Uses principles of quantum mechanics.
Quantum computers can factor in poly-time!! [Shor, 1993].
Quantum computers break most public-key cryptosystems in use!

Fake News: Quantum computers often described as efficiently solving NP problems by "trying all possibilities in parallel".
Wrong! Quantum is powerful, but doesn't efficiently solve all NP problems.

# Good Assumption for Crypto? Approximate-GCD

Greatest Common Divisor (GCD): Given integers $n_1$ and $n_2$, output their largest common factor.

# Good Assumption for Crypto? Factoring

Factoring: Given k-bit integer n, output a nontrivial factor of n.

Best-known algorithm: The "Number Field Sieve" takes $2^{O(k^{1/3} (\log k)^{1/3})}$ steps (sub-exponential in input length).

Quantum algorithm: Uses principles of quantum mechanics.
Quantum computers can factor in poly-time!! [Shor, 1993].
Quantum computers break most public-key cryptosystems in use!

Fake News: Quantum computers often described as efficiently solving NP problems by "trying all possibilities in parallel".
Wrong! Quantum is powerful, but doesn't efficiently solve all NP problems.

# Good Assumption for Crypto? Approximate-GCD

Greatest Common Divisor (GCD): Given integers $n_1$ and $n_2$, output their largest common factor.

# Good Assumption for Crypto? Approximate-GCD

Greatest Common Divisor (GCD): Given integers $n_1$ and $n_2$, output their largest common factor.

*Approximate* Greatest Common Divisor (AGCD): Given many integers $n_i = q_i \cdot p + r_i$ with $|r_i|$ much less than p, output p.

"Near-multiples" of p

Example: $r_i$ is $\lambda$ bits, p is $\lambda^2$ bits, $q_i$ is $\lambda^6$ bits. (say, $\lambda = 100$.)

Best known attacks: exponential in $\lambda$, even for quantum.

Approximate GCD, Exact Multiple Version: One of the $n_i$'s (say, $n_0$) is an exact multiple of p.

Approximate GCD, Decision Version: Decide whether the $n_i$'s are near multiples of some p, or just random integers. (Try to guess correctly more than 50% of the time.)

# A Public-Key Encryption Scheme

# Good Assumption for Crypto? Approximate-GCD

<u>Greatest Common Divisor (GCD)</u>: Given integers $n_1$ and $n_2$, output their largest common factor.

<u>*Approximate* Greatest Common Divisor (AGCD)</u>: Given many integers $n_i = q_i \cdot p + r_i$ with $|r_i|$ much less than $p$, output $p$.

"Near-multiples" of p

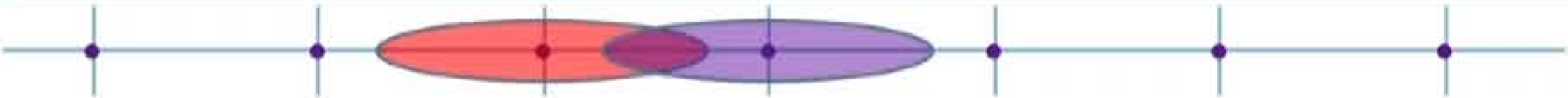
Example: $r_i$ is $\lambda$ bits, $p$ is $\lambda^2$ bits, $q_i$ is $\lambda^6$ bits. (say, $\lambda = 100$.)
Best known attacks: exponential in $\lambda$, even for quantum.

<u>Approximate GCD, Exact Multiple Version</u>: One of the $n_i$'s (say, $n_0$) is an exact multiple of p.

<u>Approximate GCD, Decision Version</u>: Decide whether the $n_i$'s are near multiples of some p, or just random integers. (Try to guess correctly more than 50% of the time.)

# A Public-Key Encryption Scheme

# Encryption Based on Approximate GCD



- Each ciphertext is a "noisy" multiple of secret integer p.
- The "noise" – the offset from the p-multiple – contains the message.
- If noise is "small", Alice recovers it as the remainder modulo p, and then recovers the message.
- If noise is too large, decryption is hopeless even for Alice.

# A Symmetric Encryption Scheme

- Shared secret key: odd number $p$
- To encrypt a bit $m$ in $\{0,1\}$:
  - Choose at random small $r$, large $q$
  - Output $c = q \cdot p + \underbrace{\boxed{2r + m}}_{}$ — the "noise"; Noise much smaller than p
    - Ciphertext is close to a multiple of p
    - m = parity of "noise" (distance to nearest multiple of p)
- To decrypt $c$:
  - Output $m = (c \bmod p) \bmod 2$

# Making It Public-Key

- Secret key is odd p (as before)
- Public key pk consists of near-multiples of p
  - Polynomially many $n_i = q_i p + 2r_i$ with $n_0$ odd
- Enc(pk, m): c ← [subset-sum($n_i$'s) + 2r + m] mod $n_0$
- Dec(sk, c): Output (c mod p) mod 2 (as before)

$c = (2\sum_{i\in S} n_i) + 2r + m - k \cdot n_0$ for some small k

$(c \ mod \ p) = (2\sum_{i\in S} r_i) + 2r + m - k \cdot 2r_0$

$(c \ mod \ p) \ mod \ 2 = m$

# Proving Security

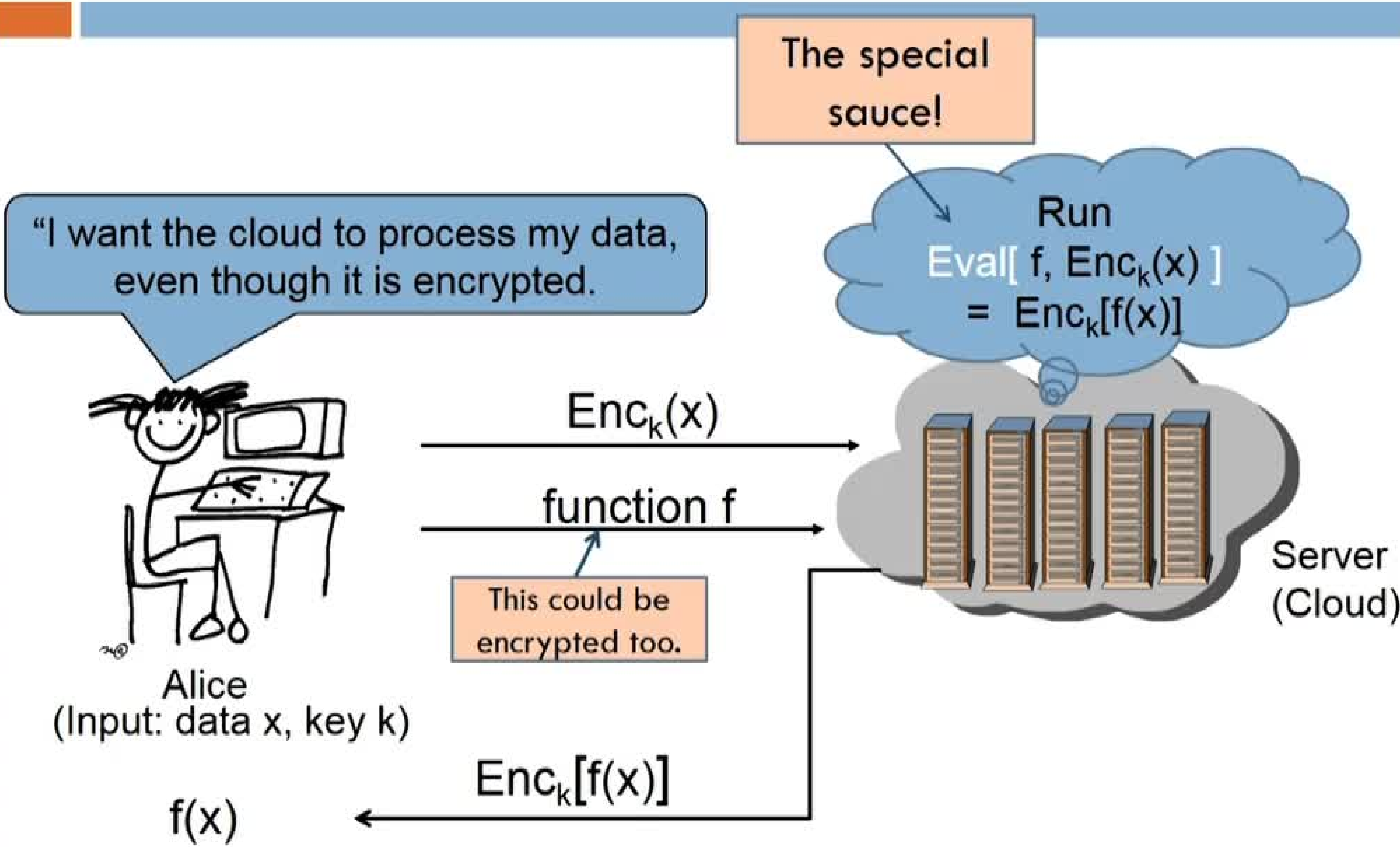
Approximate GCD, Decision Version: Decide whether integer $n_i$'s are near multiples of some p, or just random integers.

Theorem: If decision AGCD is hard, then the scheme is secure.

- Intuition:
  - Assume there is an adversary that breaks the scheme.
  - Set public key to be the integers from the AGCD problem.
  - Encrypt $m_0$ or $m_1$ with the public key.
  - If public key is well-formed (near-multiples), adversary will distinguish whether $m_0$ or $m_1$ was encrypted.
  - If public key is random (not near multiples), then the distribution Enc(pk, m) is statistically independent of m.

# Homomorphic Encryption

# Back to Homomorphic Encryption



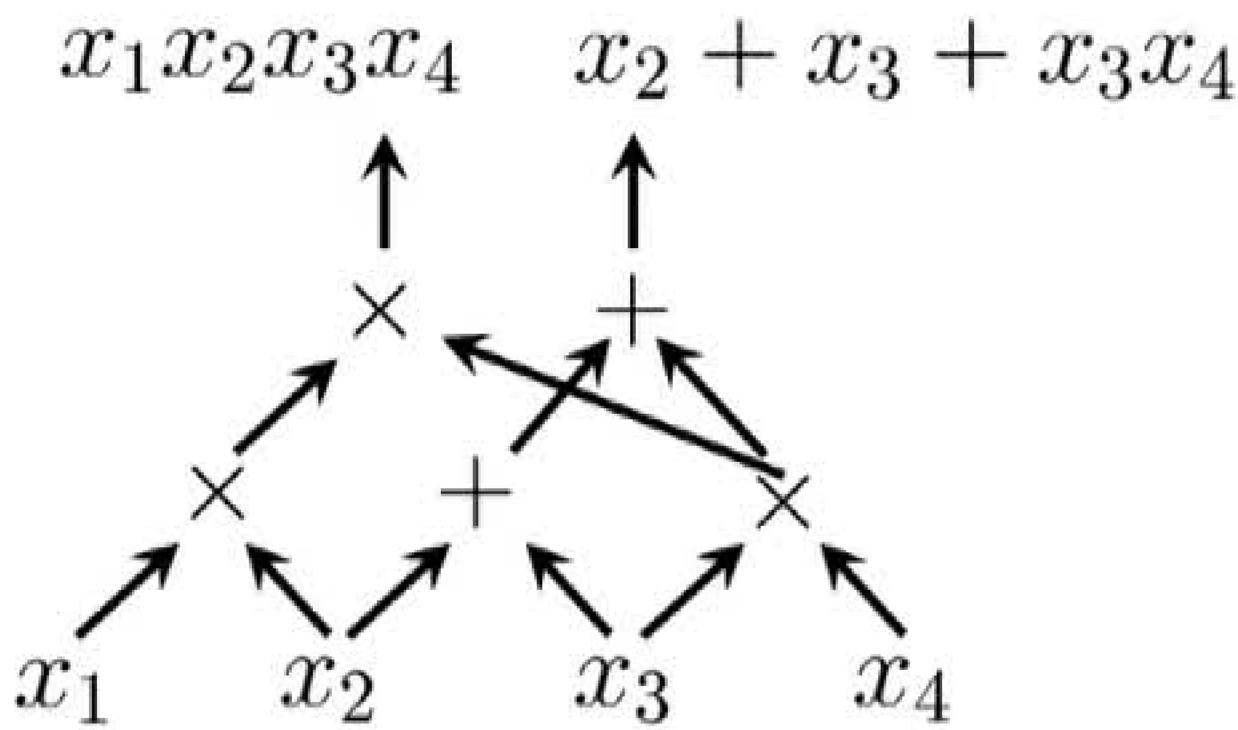
# Processing (Unencrypted) Data

- Forget encryption for a moment…
- How does your computer compute a function?
- Basically, by working on *bits*, 1's and 0's.

- Using bit operations – for example,
  - $\mathrm{AND}(b_1, b_2) = 1$ if $b_1 = b_2 = 1$; otherwise, equals $0$.
    - $\mathrm{AND}(b_1, b_2) = b_1 \times b_2$.
  - $\mathrm{XOR}(b_1, b_2) = 0$ if $b_1 = b_2$; equals $1$ if $b_1 \neq b_2$.
    - $\mathrm{XOR}(b_1, b_2) = b_1 + b_2$ (modulo 2)

# Computing General Functions

- {ADD,MULT} are Turing-complete (over any ring).
  - Take any (classically) efficiently computable function.
  - Express it as a poly-size circuit of ADD and MULT gates.




Example Circuit

# Let's Do This Encrypted…

- Let $\boxed{\mathtt{b}}$ denote a valid encryption of bit $\mathtt{b}$.
- Suppose we have a (homomorphic) encryption scheme with public functions E-ADD, E-MULT where:

  $$\texttt{E-MULT}(\boxed{\mathtt{b}_1}, \boxed{\mathtt{b}_2}) = \boxed{\mathtt{b}_1 \times \mathtt{b}_2} \quad \texttt{E-ADD}(\boxed{\mathtt{b}_1}, \boxed{\mathtt{b}_2}) = \boxed{\mathtt{b}_1 + \mathtt{b}_2}$$

  for any $\boxed{\mathtt{b}_1}$ and $\boxed{\mathtt{b}_2}$.

- Then we can ADD and MULT *encrypted* bits.
- Proceeding bit-wise, we can compute *any* function on *encrypted* data.

# Encrypted Add and Mult

Simple Idea:
Just add or multiply ciphertexts

# Let's Do This Encrypted…

- Let $\boxed{\mathtt{b}}$ denote a valid encryption of bit $\mathtt{b}$.
- Suppose we have a (homomorphic) encryption scheme with public functions E-ADD, E-MULT where:

  $$\mathtt{E\text{-}MULT}(\boxed{\mathtt{b}_1}, \boxed{\mathtt{b}_2}) = \boxed{\mathtt{b}_1 \times \mathtt{b}_2} \quad \mathtt{E\text{-}ADD}(\boxed{\mathtt{b}_1}, \boxed{\mathtt{b}_2}) = \boxed{\mathtt{b}_1 + \mathtt{b}_2}$$

  for any $\boxed{\mathtt{b}_1}$ and $\boxed{\mathtt{b}_2}$.

- Then we can ADD and MULT *encrypted* bits.
- Proceeding bit-wise, we can compute *any* function on *encrypted* data.

# Encrypted Add and Mult

Simple Idea:
Just add or multiply ciphertexts

Why should it work for our approximate gcd scheme?
If you add or multiply two near-multiples of p,
you get another near-multiple of p

# Adding and Multiplying Ciphertexts

- $c_1 = q_1 p + 2r_1 + m_1$, $c_2 = q_2 p + 2r_2 + m_2$

Noise: Distance to nearest multiple of p

- $c_1 + c_2 = (q_1 + q_2)p + \boxed{2(r_1 + r_2) + (m_1 + m_2)}$ mod $n_0 (= q_0 p)$
  - Suppose $2(r_1+r_2)+(m_1+m_2)$ is still much smaller than p
  
  ➔ $c_1+c_2$ mod $p = 2(r_1+r_2)+(m_1+m_2)$
  
  ➔ $(c_1+c_2$ mod $p)$ mod $2 = m_1+m_2$ mod $2$

Noise

- $c_1 x c_2 = (c_1 q_2 + q_1 c_2 - q_1 q_2)p + \boxed{(2r_1+m_1)(2r_2+m_2)}$ mod $n_0$
  - Suppose $(2r_1+m_1)(2r_2+m_2)$ is still much smaller than p
  
  ➔ $c_1 x c_2$ mod $p = (2r_1+m_1)(2r_2+m_2)$
  
  ➔ $(c_1 x c_2$ mod $p)$ mod $2 = m_1 x m_2$ mod $2$

# General Functions Homomorphically

- $c_1 = q_1 p + 2r_1 + m_1, \ldots, c_t = q_t p + 2r_t + m_t$
- Let f be a multivariate poly with integer coefficients (sequence of +'s and x's)
- Compute $c$ = Eval(pk, f, $c_1, \ldots, c_t) = f(c_1, \ldots, c_t) \bmod n_0$

  Suppose this noise is much smaller than p

  - $f(c_1, \ldots, c_t) = \boxed{f(2r_1+m_1, \ldots, 2r_t+m_t)} + qp$
  - Then $(c \bmod p) \bmod 2 = f(m_1, \ldots, m_t) \bmod 2$

That's what we want!

Problem: Noise grows exponentially with f's degree

# Wait – Why Bother with Noise at all?

- Try to use ring homomorphisms (without noise)
  - Ciphertexts and messages live in rings $R_C$ and $R_M$.
  - Decryption is a ring homomorphism $D : R_C \rightarrow R_M$.
  - Homomorphic ops $+$ and $\times$ on ciphertexts in $R_C$ induce $+$ and $\times$ on messages in $R_M$.
  - Security: Encryptions of 0 form an ideal in $R_C$. Secure only if "ideal membership problem" is hard.

# Wait – Why Bother with Noise at all?

- Example [Polly Cracker by Fellows & Koblitz]:
  - Encryptions of m are polynomials that evaluate to m at some secret point s.
- Attacking Polly Cracker:
  - Case 1: The multivariate ciphertext polynomials can be represented over a polynomial-size monomial basis.
    - Ideal membership problem is easy. Solve using linear algebra.
  - Case 2: Well then how are ciphertext polynomials represented? (Ciphertexts must be compact.)

# Bootstrapping: A Way to Refresh Noisy Ciphertexts

# A Digression into Philosophy…

- Can the human mind understand itself?
  - Or, as a mind becomes more complex, does the task of understanding also become more complex, so that self-understanding it always just out of reach?

- Self-reference often causes problems, even in mathematics and CS
  - Godel's incompleteness theorem
  - Turing's Halting Problem

# Philosophy Meets Cryptography

- Can a homomorphic encryption scheme decrypt itself?
  - If we run Eval(pk, Dec(·,·), $c_1$, …, $c_t$), does it work?
  - Suppose our HE scheme can Eval depth-d circuits:
    - Is it always true that HE's Dec function has depth > d?
    - Is Dec(·,·) always just beyond the Eval capacity of the HE scheme?

- Bootstrapping: the process of running Eval(·,…,·) on Dec(·,·)

# Bootstrapping: What Is It?

- So far, we can evaluate bounded depth funcs F:

# Bootstrapping: What Is It?

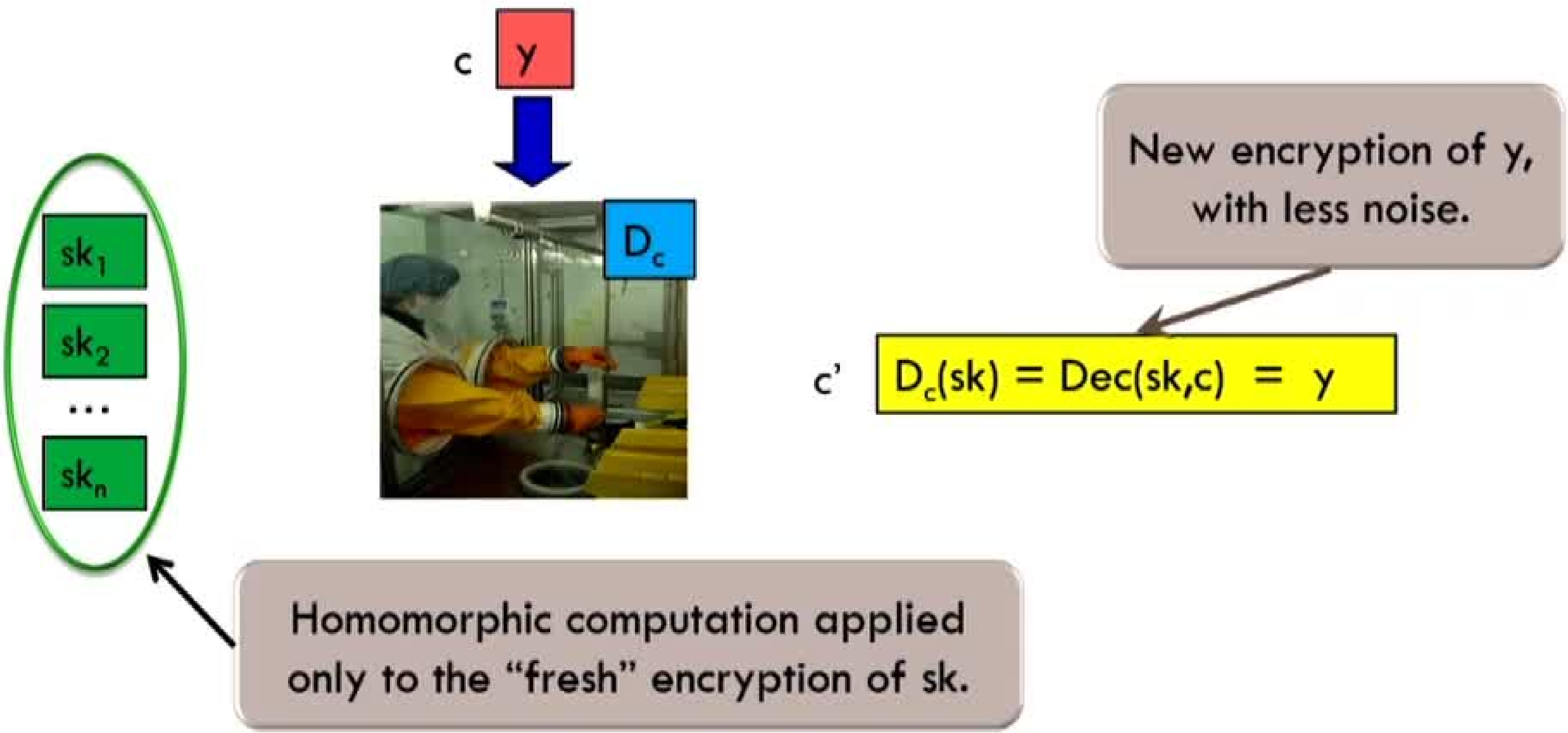
- So far, we can evaluate bounded depth funcs F:


$f(m_1, m_2, \ldots, m_t)$

- We have a noisy evaluated ciphertext c.
- We want to get a less noisy c' that encrypts the same value, but with less noise.
- Bootstrapping *refreshes* ciphertexts, using the *encrypted secret key*.

# Bootstrapping: What Is It?

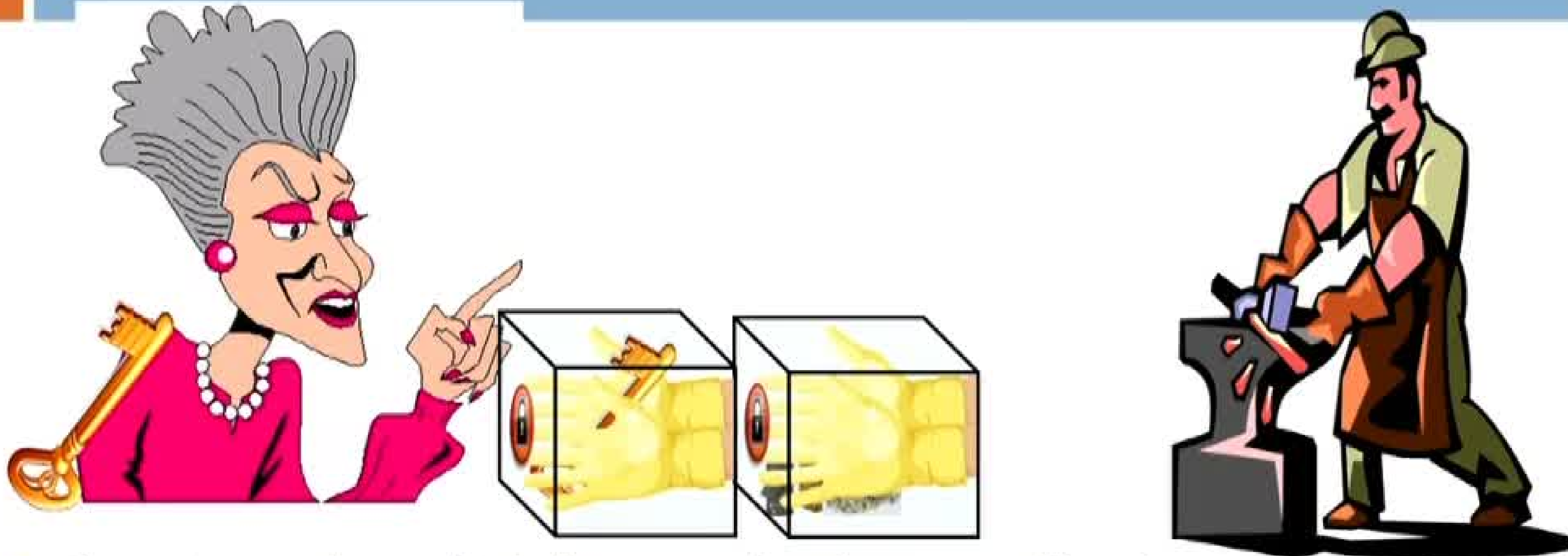
- For ciphertext c, consider $D_c(sk) = Dec(sk,c)$
  - Suppose $D_c(\cdot)$ is a low-degree polynomial in sk.
- Include in the public key also $Enc_{pk}(sk)$.



# Bootstrappable Schemes

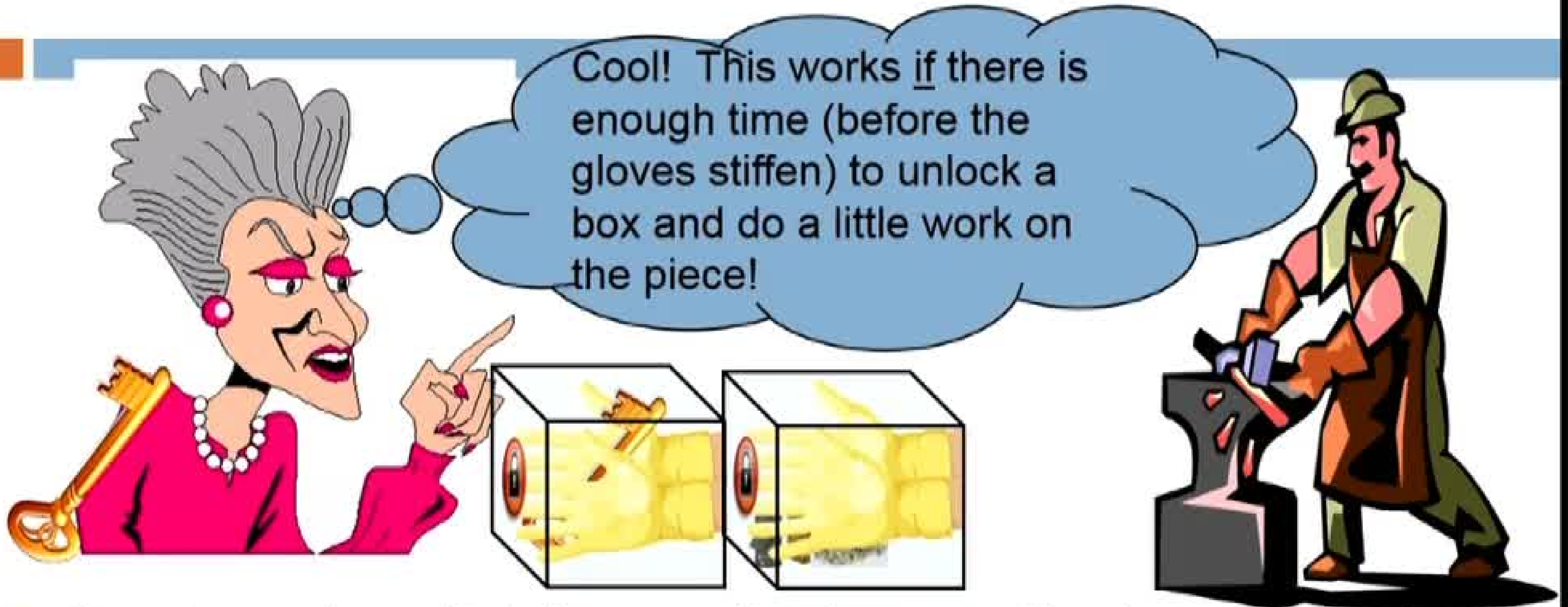
- Bootstrappable HE $\longrightarrow$ Fully Homomorphic Encryption
- Can our integer-based HE scheme be bootstrapped?
  - Yes, after some tweaks.
- Known FHE schemes all use similar techniques
  - All use noise
  - All use bootstrapping
  - All rely on hardness of "lattice" problems

# A Physical Analogy for Bootstrapping



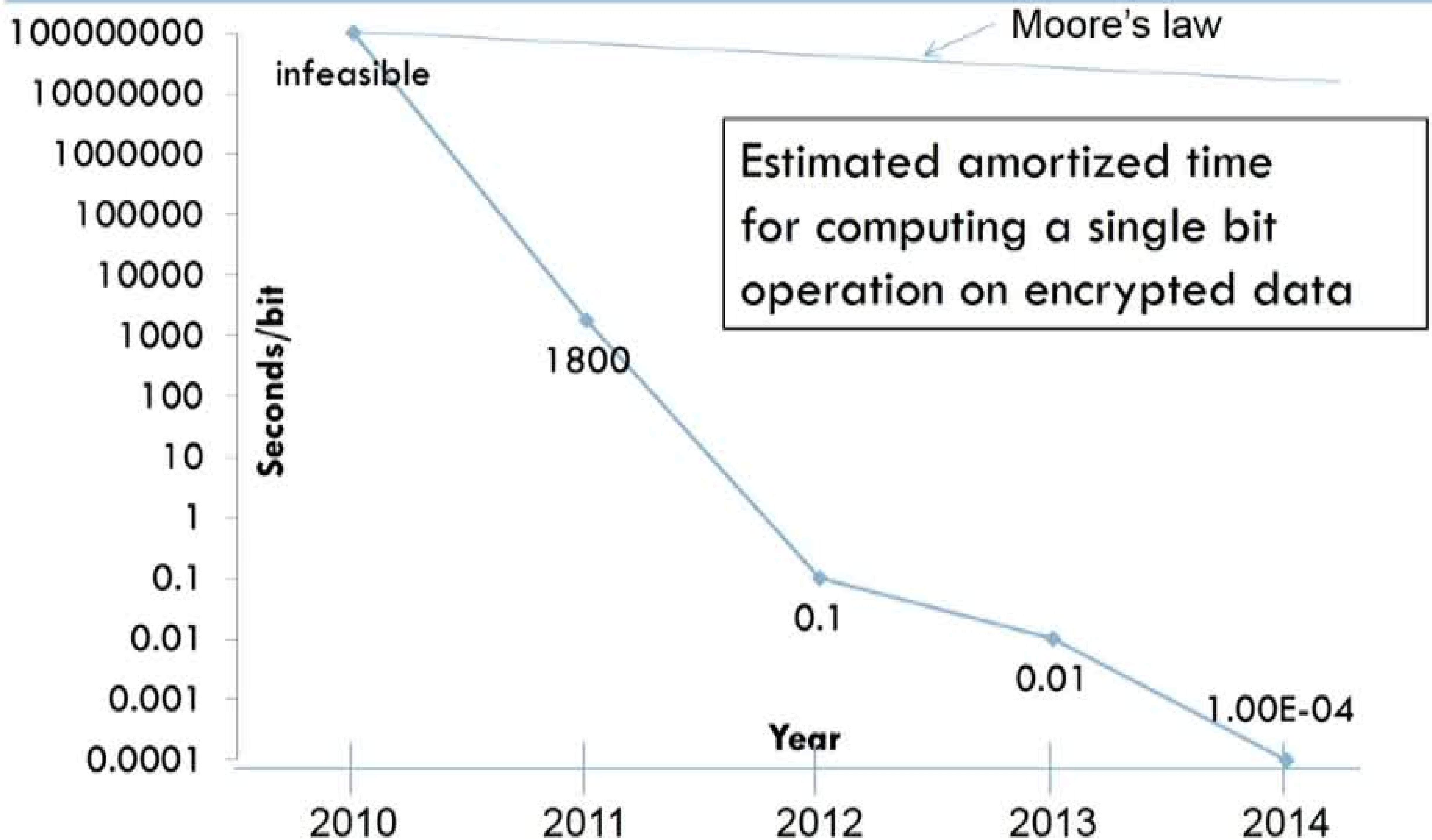
- Alice gives worker multiple boxes, each with a copy of her key inside
- Worker assembles jewel inside box #1 for 1 minute.
- Then, worker puts box #1 inside box #2!
- With box #2's gloves, worker opens box #1 with key, takes jewel out, and continues assembling till box #2's gloves stiffen.
- And so on…

# A Physical Analogy for Bootstrapping

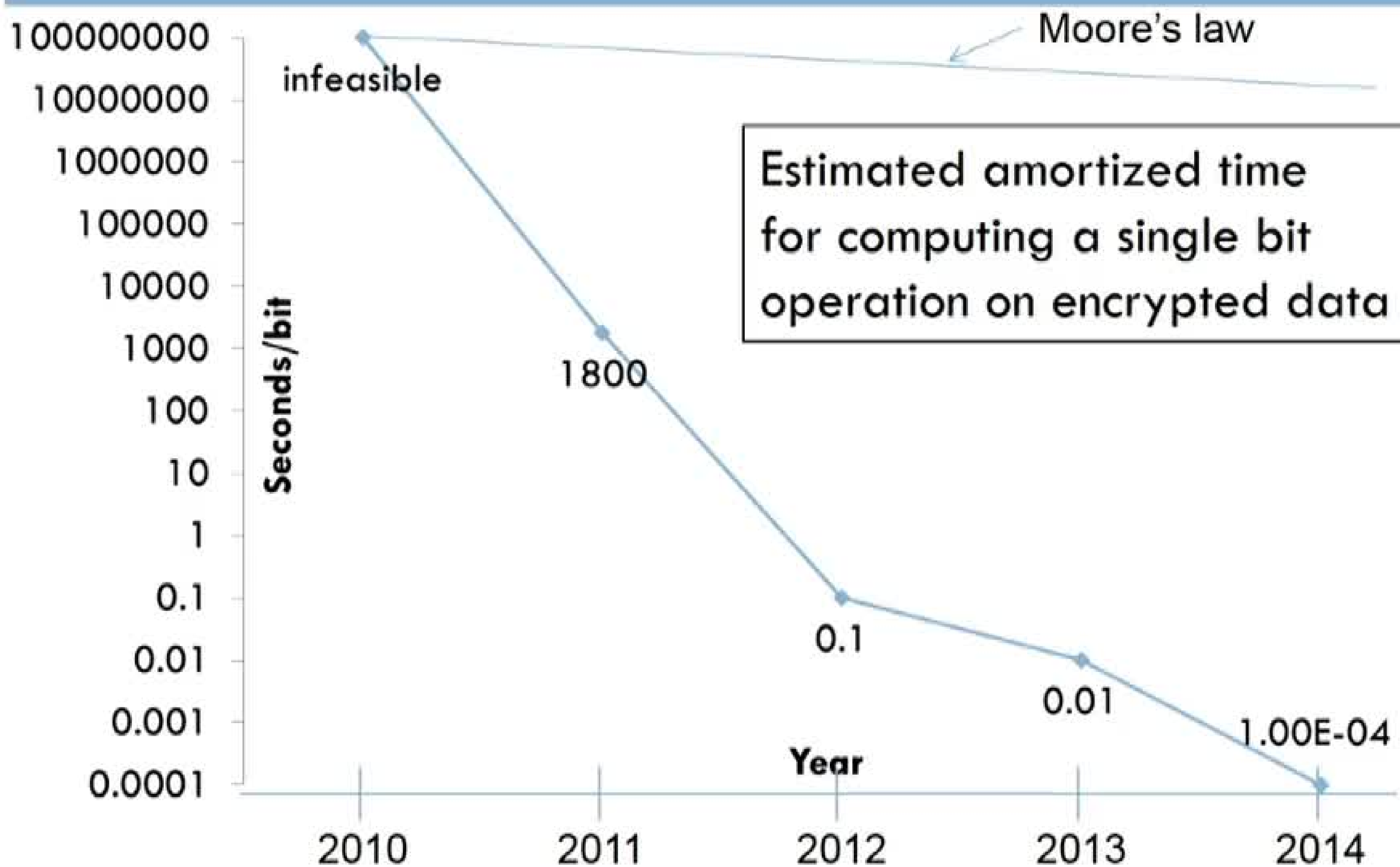



- Alice gives worker multiple boxes, each with a copy of her key inside
- Worker assembles jewel inside box #1 for 1 minute.
- Then, worker puts box #1 inside box #2!
- With box #2's gloves, worker opens box #1 with key, takes jewel out, and continues assembling till box #2's gloves stiffen.
- And so on…

# Speed of Computing on Encrypted Data on IBM's HElib Platform



# Speed of Computing on Encrypted Data on IBM's HElib Platform




Example: Can compare two genome sequences with ~100,000 SNPs in 5-10 minutes