Fully Homomorphic Encryption over the Integers

Many slides borrowed from Craig

Marten van Dijk[1], Craig Gentry[2], Shai Halevi[2], Vinod Vaikuntanathan[2]

1 – MIT, 2 – IBM Research

The Goal

I want to delegate processing of my data, without giving away access to it.

Application: Cloud Computing

I want to delegate processing of my data, without giving away access to it.

- Storing my files on the cloud
  - Encrypt them to protect my information
  - Later, I want to retrieve the files containing "cloud" within 5 words of "computing".
    - Cloud should return only these (encrypted) files, without knowing the key

Computing on Encrypted Data

- Separating processing from access via encryption:
  - I will encrypt my stuff before sending it to the cloud
  - They will apply their processing on the encrypted data, send me back the processed result
  - I will decrypt the result and get my answer

Application: Private Google Search

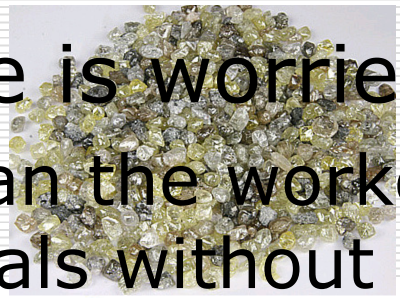
> I want to delegate processing of my data, without giving away access to it.

- Private Internet search
  - Encrypt my query, send to Google
    - Google cannot "see" my query, since it does not know my key
  - I still want to get the same results
    - Results would be encrypted too
- Privacy combo: Encrypted query on encrypted data

An Analogy: Alice’s Jewelry Store

- Alice’s workers need to assemble raw materials into jewelry
- But Alice is worried about theft

  How can the workers process the raw materials without having access to them?



An Analogy: Alice's Jewelry Store

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

The Analogy

- Encrypt: putting things inside the box
  - Anyone can do this (imagine a mail-drop)
  - $c_i \leftarrow \text{Enc}(m_i)$
- Decrypt: Taking things out of the box
  - Only Alice can do it, requires the key
  - $m^* \leftarrow \text{Dec}(c^*)$
- Process: Assembling the jewelry
  - Anyone can do it, computing on ciphertext
  - $c^* \leftarrow \text{Process}(c_1,\ldots,c_n)$
- $m^* = \text{Dec}(c^*)$ is "the ring", made from "raw materials" m_i

Public-key Encryption

- Three procedures: KeyGen, Enc, Dec
  - (sk,pk) ← KeyGen($)
    - Generate random public/secret key-pair
  - c ← $Enc_{pk}(m)$
    - Encrypt a message with the public key
  - m ← $Dec_{sk}(c)$
    - Decrypt a ciphertext with the secret key
- E.g., RSA: $c \leftarrow m^e \bmod N$, $m \leftarrow c^d \bmod N$
  - (N,e) public key, d secret key

Homomorphic Public-key Encryption

- Another procedure: Eval (for Evaluate)
  - $c^* \leftarrow \text{Eval}(pk, f, c_1,\ldots,c_t)$

Encryption of $f(m_1,\ldots,m_t)$.
I.e., $\text{Dec}(sk, c) = f(m_1, \ldots m_t)$

function

Encryptions of inputs $m_1,\ldots,m_t$ to f

  - No info about $m_1, \ldots, m_t$, $f(m_1, \ldots m_t)$ is leaked
  - $f(m_1, \ldots m_t)$ is the "ring" made from raw materials $m_1, \ldots, m_t$ inside the encryption box

Can we do it?

- As described so far, sure..
  - $(\Pi, c_1,\ldots,c_n) = c^* \leftarrow \mathrm{Eval}_{pk}(\Pi, c_1,\ldots,c_n)$
  - $\mathrm{Dec}_{sk}(c^*)$ decrypts individual c_i's, apply Π

(the workers do nothing, Alice assembles the jewelry by herself)

This is the main challenge

Of course, this is cheating:

- We want c^* to remain small
  - independent of the size of Π
  - "Compact" homomorphic encryption
- We may also want Π to remain secret

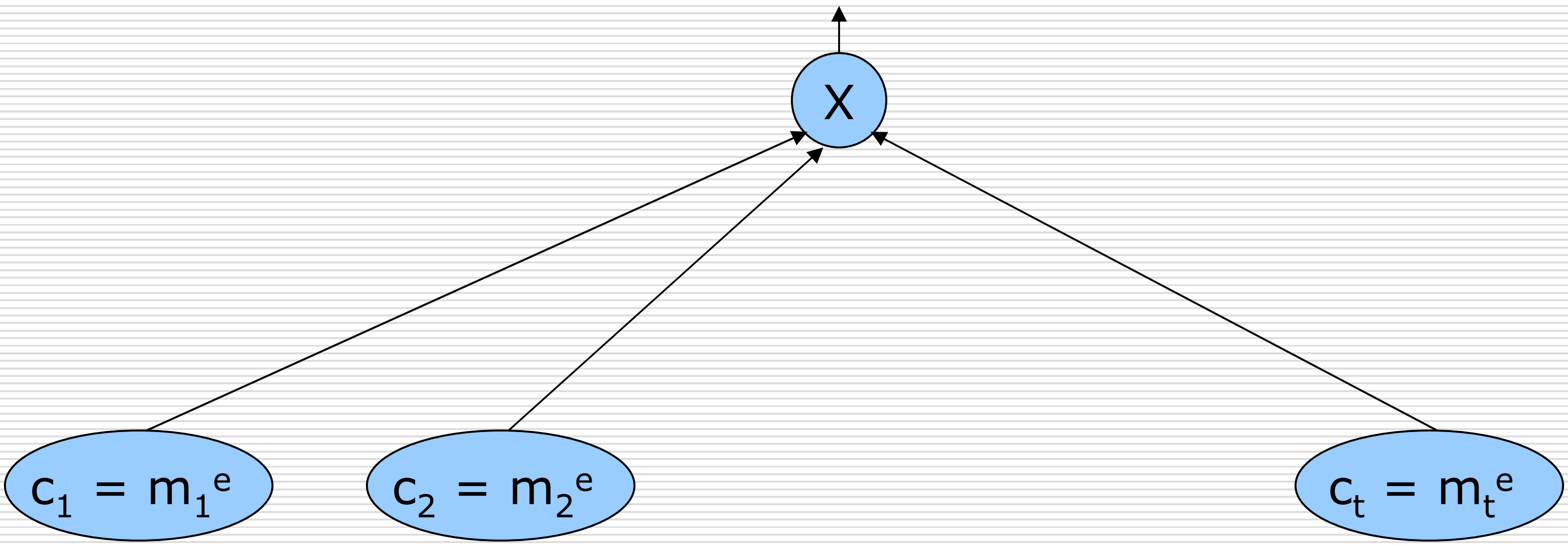
Can be done with "generic tools" (Yao's garbled circuits)

Previous Schemes

$c \leftarrow \text{Eval}(pk, f, c_1,\ldots,c_t)$,
$\text{Dec}(sk, c) = f(m_1, \ldots, m_t)$

- Only "somewhat homomorphic"
  - Can only handle some functions f
- RSA works for MULT function (mod N)

$c = c_1 \times \ldots \times c_t = (m_1 \times \ldots \times m_t)^e \pmod N$



“Somewhat Homomorphic” Schemes

- RSA, ElGamal work for MULT mod N
- GoMi, Paillier work for XOR, ADD
- BGN05 works for quadratic formulas

Schemes with large ciphertext

- SYY99 works for shallow fan-in-2 circuits
  - c* grows exponentially with the depth of f
- IsPe07 works for branching program
  - c* grows with length of program
- AMGH08 for low-degree polynomials
  - c* grows exponentially with degree

Connection with 2-party computation

- Can get "homomorphic encryption" from certain protocols for 2-party secure function evaluation
  - E.g., Yao86
- But size of c*, complexity of decryption, more than complexity of the function f
  - Think of Alice assembling the ring herself
- These are solving a different problem

A Recent Breakthrough

- Genrty09: A bootstrapping technique

- Gentry also described a candidate "bootstrappable" scheme
  - Based on ideal lattices

The Current Work

- A second “bootstrappable” scheme
  - Very simple: using only modular arithmetic
- Security is based on the hardness of finding “approximate-GCD”

Outline

1. Homomorphic symmetric encryption
   - Very simple
2. Turning it into public-key encryption
   - Result is “almost bootstrappable”
3. Making it bootstrappable
   - Similar to Gentry’09
4. Security

   As much as we have time
5. Gentry’s bootstrapping technique

   Not today

A homomorphic symmetric encryption

- Shared secret key: odd number p
- To encrypt a bit m:
  - Choose at random small r, large q
  - Output $c = m + 2r + pq$ (The "noise": $m + 2r$; Noise much smaller than p)
    - Ciphertext is close to a multiple of p
    - m = LSB of distance to nearest multiple of p
- To decrypt c:
  - Output $m = (c \bmod p) \bmod 2$
    - $m = c - p \cdot [c/p] \bmod 2$
      $= c - [c/p] \bmod 2$
      $= \text{LSB}(c)\ \text{XOR}\ \text{LSB}([c/p])$

Homomorphic Public-Key Encryption

- Secret key is an odd p as before
- Public key is many "encryptions of 0"
  - $x_i = [q_i p + 2r_i]_{x0}$ for i=1,2,…,t
- $\text{Enc}_{pk}(m) = [\text{subset-sum}(x_i\text{'s})+m]_{x0}$
- $\text{Dec}_{sk}(c) = (c \bmod p) \bmod 2$

Why is this homomorphic?

- Basically because:
  - If you add or multiply two near-multiples of p, you get another near multiple of p…

Why is this homomorphic?

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

Distance to nearest multiple of p

- $c_1 + c_2 = (q_1 + q_2)p + 2(r_1 + r_2) + (m_1 + m_2)$
  - $2(r_1 + r_2) + (m_1 + m_2)$ still much smaller than p

  $\rightarrow c_1 + c_2 \bmod p = 2(r_1 + r_2) + (m_1 + m_2)$

- $c_1 \times c_2 = (c_1 q_2 + q_1 c_2 - q_1 q_2)p + 2(2r_1 r_2 + r_1 m_2 + m_1 r_2) + m_1 m_2$
  - $2(2r_1 r_2 + \ldots)$ still much smaller than p

  $\rightarrow c_1 x c_2 \bmod p = 2(2r_1 r_2 + \ldots) + m_1 m_2$

Why is this homomorphic?

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

  Suppose this noise is much smaller than p

  - $f(c_1, \ldots, c_t) = f(m_1+2r_1, \ldots, m_t+2r_t) + qp$
    $= f(m_1, \ldots, m_t) + 2r \quad + qp$
  - Then (c mod p) mod 2 = $f(m_1, \ldots, m_t)$ mod 2

That's what we want!

How homomorphic is this?

- Can keep adding and multiplying until the "noise term" grows larger than p/2
  - Noise doubles on addition, squares on multiplication
  - Multiplying d ciphertexts ➔ noise of size $\sim 2^{dn}$
- We choose $r \sim 2^n$, $p \sim 2^{n^2}$ (and $q \sim 2^{n^5}$)
  - Can compute polynomials of degree n before the noise grows too large

Keeping it small

- The ciphertext's bit-length doubles with every multiplication
  - The original ciphertext already has n^6 bits
  - After ~log n multiplications we get $\sim n^7$ bits
- We can keep the bit-length at n^6 by adding more "encryption of zero"
  - $|y_1|=n^6+1$, $|y_2|=n^6+2$, …, $|y_m|=2n^6$
  - Whenever the ciphertext length grows, set $c' = c \bmod y_m \bmod y_{m-1} \ldots \bmod y_1$

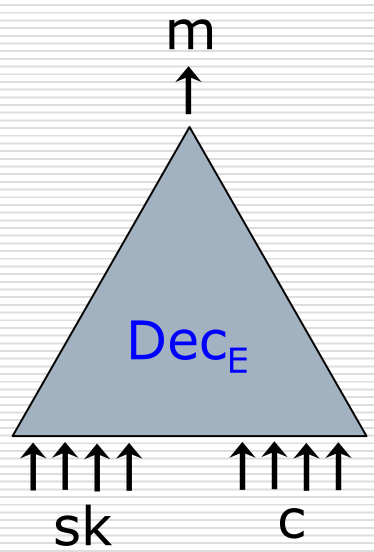
Bootstrappable yet?

c/p, rounded to nearest integer

- Almost, but not quite:
- Decryption is $m = LSB(c) \oplus LSB([c/p])$
  - Computing [c/p] takes degree O(n)
  - But O() is more than one (maybe 7??)
    - Integer c has $\sim n^5$ bits
  - Our scheme only supports degree $\leq n$
- To get a bootstrappable scheme, use Gentry09 technique to "squash the decryption circuit"

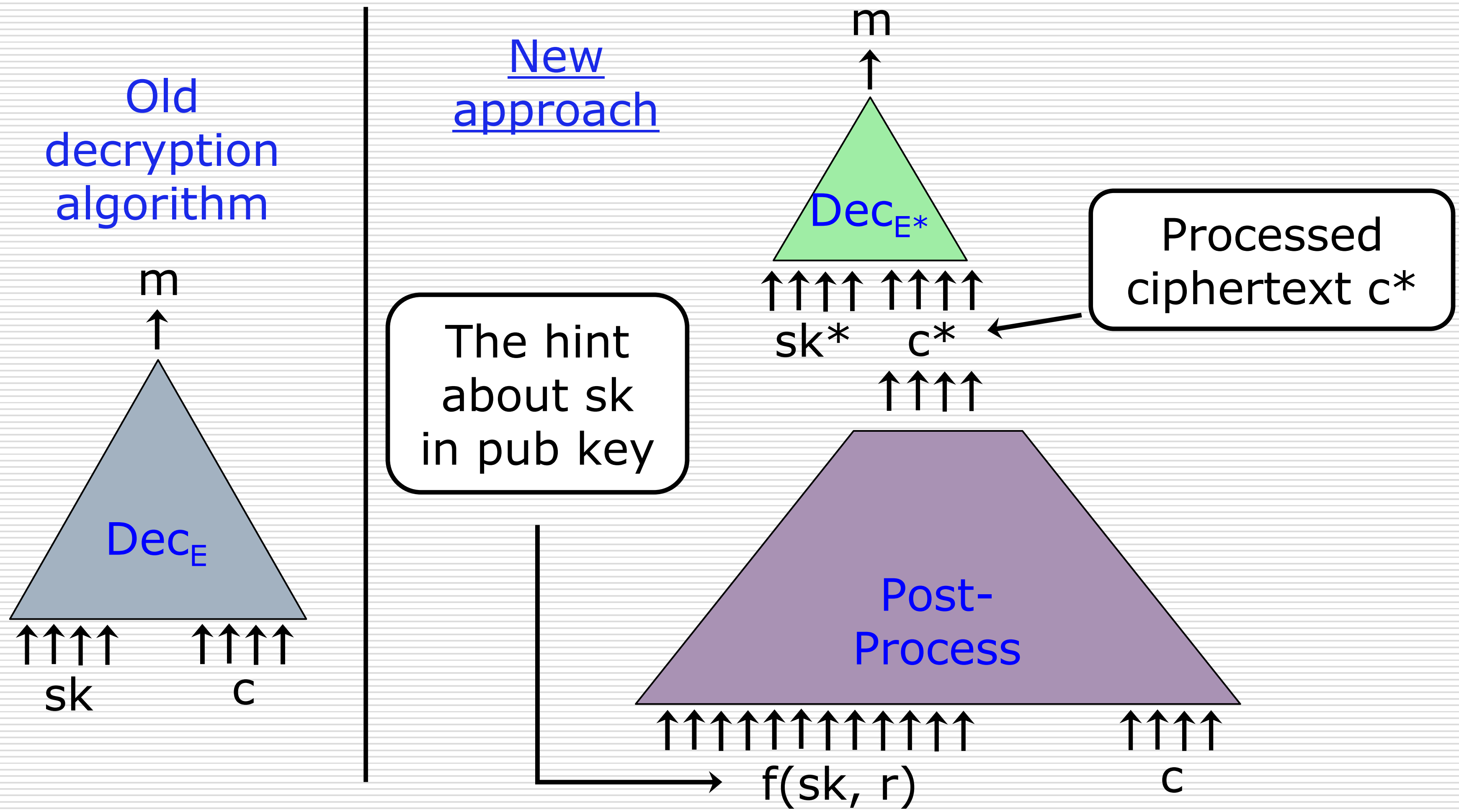
How do we "simplify" decryption?

Old decryption algorithm



- Idea: Add to public key another "hint" about sk
  - Of course, hint should not break secrecy of encryption
- With hint, anyone can post-process the ciphertext, leaving less work for Dec_{E*} to do
- This idea is used in server-aided cryptography.

How do we “simplify” decryption?



Hint in pub key lets anyone post-process the ciphertext, leaving less work for Dec_{E*} to do.

Squashing the decryption circuit

- Add to public key many real numbers
  - $d_1, d_2, \ldots, d_t \in [0,2]$ (with "sufficient precision")
  - $\exists$ sparse set S for which $\Sigma_{i\in S}\, d_i = 1/p \bmod 2$
- Enc, Eval output $\psi_i = c \times d_i \bmod 2$, $i=1,\ldots,t$
  - Together with c itself
- New secret key is bit-vector $\sigma_1,\ldots,\sigma_t$
  - $\sigma_i=1$ if $i\in S$, $\sigma_i=0$ otherwise
- New Dec(c) is $c - [\Sigma_i\, \sigma_i \Psi_i] \bmod 2$
  - Can be computed with a "low-degree circuit" because S is sparse

A Different Way to Add Numbers

- $\text{Dec}_{E*}(s,c) = \text{LSB}(c) \text{ XOR } \text{LSB}([\Sigma_i \overbrace{\sigma_i \psi_i}^{a_i}])$

a_i's in binary representation

$a_{1,0}$	$a_{1,-1}$	…	$a_{1,-\log t}$
$a_{2,0}$	$a_{2,-1}$	…	$a_{2,-\log t}$
$a_{3,0}$	$a_{3,-1}$	…	$a_{3,-\log t}$
$a_{4,0}$	$a_{4,-1}$	…	$a_{4,-\log t}$
$a_{5,0}$	$a_{5,-1}$	…	$a_{5,-\log t}$
…	…	…	…
$a_{t,0}$	$a_{t,-1}$	…	$a_{t,-\log t}$

Our problem: t is large (e.g. n^6)

A Different Way to Add Numbers

Let b_0 be the binary rep of Hamming weight

			$a_{1,0}$	$a_{1,-1}$	…	$a_{1,-\log t}$
			$a_{2,0}$	$a_{2,-1}$	…	$a_{2,-\log t}$
			$a_{3,0}$	$a_{3,-1}$	…	$a_{3,-\log t}$
			$a_{4,0}$	$a_{4,-1}$	…	$a_{4,-\log t}$
			$a_{5,0}$	$a_{5,-1}$	…	$a_{5,-\log t}$
			…	…	…	…
			$a_{t,0}$	$a_{t,-1}$	…	$a_{t,-\log t}$
$b_{0,\log t}$	…	$b_{0,1}$	$b_{0,0}$			

A Different Way to Add Numbers

Let b_{-1} be the binary rep of Hamming weight

$a_{1,0}$	$a_{1,-1}$	…	$a_{1,-\log t}$
$a_{2,0}$	$a_{2,-1}$	…	$a_{2,-\log t}$
$a_{3,0}$	$a_{3,-1}$	…	$a_{3,-\log t}$
$a_{4,0}$	$a_{4,-1}$	…	$a_{4,-\log t}$
$a_{5,0}$	$a_{5,-1}$	…	$a_{5,-\log t}$
…	…	…	…
$a_{t,0}$	$a_{t,-1}$	…	$a_{n,-\log t}$

$b_{0,\log t}$	…	$b_{0,1}$	$b_{0,0}$			
	$b_{-1,\log t}$	…	$b_{-1,1}$	$b_{-1,0}$		

A Different Way to Add Numbers

Let $b_{-\log t}$ be the binary rep of Hamming weight

$a_{1,0}$	$a_{1,-1}$	…	$a_{1,-\log t}$
$a_{2,0}$	$a_{2,-1}$	…	$a_{2,-\log t}$
$a_{3,0}$	$a_{3,-1}$	…	$a_{3,-\log t}$
$a_{4,0}$	$a_{4,-1}$	…	$a_{4,-\log t}$
$a_{5,0}$	$a_{5,-1}$	…	$a_{5,-\log t}$
…	…	…	…
$a_{t,0}$	$a_{t,-1}$	…	$a_{t,-\log t}$

$b_{0,\log t}$	…	$b_{0,1}$	$b_{0,0}$			
	$b_{-1,\log t}$	…	$b_{-1,1}$	$b_{-1,0}$		
		…	…	…	…	
			$b_{-\log t,\log t}$	…	$b_{-\log t,1}$	$b_{-\log t,0}$

A Different Way to Add Numbers

Only log t numbers with log t bits of precision. Easy to handle.

$a_{1,0}$	$a_{1,-1}$	…	$a_{1,-\log t}$
$a_{2,0}$	$a_{2,-1}$	…	$a_{2,-\log t}$
$a_{3,0}$	$a_{3,-1}$	…	$a_{3,-\log t}$
$a_{4,0}$	$a_{4,-1}$	…	$a_{4,-\log t}$
$a_{5,0}$	$a_{5,-1}$	…	$a_{5,-\log t}$
…	…	…	…
$a_{t,0}$	$a_{t,-1}$	…	$a_{n,-\log t}$

$b_{0,\log t}$	…	$b_{0,1}$	$b_{0,0}$			
	$b_{-1,\log t}$	…	$b_{-1,1}$	$b_{-1,0}$		
		…	…	…	…	
			$b_{-\log n,\log t}$	…	$b_{-\log t,1}$	$b_{-\log t,0}$

Computing Sparse Hamming Wgt.

$a_{1,0}$	$a_{1,-1}$	…	$a_{1,-\log n}$
$a_{2,0}$	$a_{2,-1}$	…	$a_{2,-\log n}$
$a_{3,0}$	$a_{3,-1}$	…	$a_{3,-\log n}$
$a_{4,0}$	$a_{4,-1}$	…	$a_{4,-\log n}$
$a_{5,0}$	$a_{5,-1}$	…	$a_{5,-\log n}$
…	…	…	…
$a_{t,0}$	$a_{t,-1}$	…	$a_{t,-\log t}$

Computing Sparse Hamming Wgt.

$a_{1,0}$	$a_{1,-1}$	…	$a_{1,-\log t}$
0	0	…	0
0	0	…	0
$a_{4,0}$	$a_{4,-1}$	…	$a_{4,-\log t}$
0	0	…	0
…	…	…	…
$a_{t,0}$	$a_{t,-1}$	…	$a_{t,-\log t}$

Computing Sparse Hamming Wgt.

- Binary representation of the Hamming weight of $\mathbf{a} = (a_1, \ldots, a_t) \in \{0,1\}^t$
  - The i'th bit of HW($\mathbf{a}$) is $e_{2^i}(\mathbf{a})$ mod2
  - e_k is elementary symmetric poly of degree k
    - Sum of all products of k bits
- We know *a priori* that weight ≤ |S|
  - ➔ Only need upto $e_{2^\wedge[\log |S|]}(\mathbf{a})$
  - ➔ Polynomials of degree upto |S|
- Set |S| ~ n, then E* is bootstrappable.

Security

- The approximate-GCD problem:
  - Input: integers $w_0, w_1, \dots, w_t$,
    - Chosen as $w_i = q_i p + r_i$ for a secret odd p
    - $p \in_\$ [0,P]$, $q_i \in_\$ [0,Q]$, $r_i \in_\$ [0,R]$ (with $R \ll P \ll Q$)
  - Task: find p
- Thm: If we can distinguish Enc(0)/Enc(1) for some p, then we can find that p
  - Roughly: the LSB of r_i is a "hard core bit"
  - ➔ Scheme is secure if approx-GCD is hard
- Is approx-GCD really a hard problem?

Hard-core-bit theorem

A. The approximate-GCD problem:

- Input: $w_i = q_i p + r_i$ ($i=0,\ldots,t$)
  - $p \in_\$ [0,P]$, $q_i \in_\$ [0,Q]$, $r_i \in_\$ [0,R']$ (with $R' \ll P \ll Q$)
- Task: find p

B. The cryptosystem

- Input: $x_i = q_i p + 2r_i$ ($i=0,\ldots,t$), $c=qp+2r+m$
  - $p \in_\$ [0,P]$, $q_i \in_\$ [0,Q]$, $r_i \in_\$ [0,R]$ (with $R \ll P \ll Q$)
- Task: distinguish $m=0$ from $m=1$

- Thm: Solution to B ➔ solution to A
  - small caveat: R′ smaller than R

Proof outline

- Input: $w_i = q_i p + r_i$ $(i=1,\ldots,t)$
- Use the w_i's to form a public key
  - This is where we need $R' > R$
- Amplify the distinguishing advantage
  - From any noticeable ε to almost 1
- Use reliable distinguisher to learn q_t
  - Using the binary GCD procedure
- Finally $p = \text{round}(w_t/q_t)$

Use the w_i's to form a public key

- We have $w_i=q_ip+r_i$, need $x_i=q_i'p+2r_i'$
  - Setting $x_i = 2w_i$ yields wrong distribution
- Reorder w_i's so w_0 is the largest one
  - Check that w_0 is odd, else abort
  - Also hope that q_0 is odd (else may fail to find p)
    - w_0 odd, q_0 odd → r_0 is even
- $x_0=w_0+2\rho_0$, $x_i=(2w_i+2\rho_i) \bmod w_0$ for $i>0$
  - The ρ_i's are random < R
- Correctness:
  1. $r_i+\rho_i$ distributed almost identically to ρ_i
     - Since $R>R'$ by a super-polynomial factor
  2. $2q_i \bmod q_0$ is random in $[q_0]$

Amplify the distinguishing advantage

- Given an integer $z=qp+r$, with $r<R'$:

  Set $c = [z+ m+2\rho + \text{subset-sum}(x_i\text{'s})] \bmod x_0$
  - For random $\rho<R$, random bit m
- c is a random ciphertext wrt the x_i's
  - $\rho>r_i$'s, so $\rho+r_i$'s distributed like ρ
  - (subset-sum(q_i)'s mod q_0) random in $[q_0]$
- $c \bmod p \bmod 2 = r+m \bmod 2$
  - A guess for $c \bmod p \bmod 2$ → vote for $r \bmod 2$
- Choose many random c's, take majority
  - Noticeable advantage → Reliable $r \bmod 2$

Use reliable distinguisher to learn $q_{t'}$

- From $z=qp+r$, can get $r \bmod 2$
  - Note: $z = q+r \bmod 2$ (since p is odd)
  - So $(q \bmod 2) = (r \bmod 2) \oplus (z \bmod 2)$
- Given z_1, z_2, both near multiples of p

Binary-GCD

  - Get $b_i := q_i \bmod 2$, if $z_1<z_2$ swap them
  - If $b_1=b_2=1$, set $z_1:=z_1-z_2$, $b_1:=b_1-b_2$
    - At least one of the b_i's must be zero now
  - For any $b_i=0$ set $z_i := \text{floor}(z_i/2)$
    - new-q_i = old-$q_i/2$
  - Repeat until one z_i is zero, output the other

$z = (2s)p + r \rightarrow z/2 = sp + r/2$

$\rightarrow \text{floor}(z/2) = sp + \text{floor}(r/2)$

Use reliable distinguisher to learn q_t

- $z_i = q_i p + r_i$, $i=1,2$, $z' := \text{Binary-GCD}(z_1, z_2)$
  - Then $z' = \text{GCD}^*(q_1, q_2) \cdot p + r'$ (The odd part of the GCD)
  - For random q_1, q_2, $\Pr[\text{GCD}(q_1, q_2)=1] \sim 0.6$
- Try (say) $z' := \text{Binary-GCD}(w_t, w_{t-1})$
  - Hope that $z' = 1 \cdot p + r$
    - Else try again with $\text{Binary-GCD}(z', w_{t-2})$, etc.
- Run $\text{Binary-GCD}(w_t, z')$
  - The b_2 bits spell out the bits of q_t
- Once you learn q_t then
  - $\text{round}(w_t/q_t) = p + \text{round}(r_t/q_t) = p$

Hardness of Approximate-GCD

- Several lattice-based approaches for solving approximate-GCD
  - Related to Simultaneous Diophantine Approximation (SDA)
  - Studied in [Hawgrave-Graham01]
    - We considered some extensions of his attacks
- All run out of steam when $|q_i|>|p|^2$
  - In our case $|p|\sim n^2$, $|q_i|\sim n^5 \gg |p|^2$

Relation to SDA

- $x_i = q_i p + r_i$ ($r_i \ll p \ll q_i$), $i = 0,1,2,\ldots$
  - $y_i = x_i/x_0 = (q_i p + r_i)/(q_0 p + r_0)$
    $= (q_i + (r_i/p))/(q_0 + (r_0/p))$
    - $= (q_i+s_i)/q_0$, with $s_i \sim r_i/p \ll 1$
  - y1, y2, … is an instance of SDA
    - q_0 is a denominator that approximates all y_i's
- Use Lagarias'es algorithm to try and solve this SDA instance
  - Find q_0, then $p=\mathrm{round}(x_0/q_0)$

Lagarias'es SDA algorithm

- Consider the rows of this matrix B:
  - They span dim-(t+1) lattice

$$B=\begin{pmatrix} R & x_1 & x_2 & \dots & x_t \\ & -x_0 & & & \\ & & -x_0 & & \\ & & & \dots & \\ & & & & -x_0 \end{pmatrix}$$

- $\langle q_0,q_1,\dots,q_t\rangle\cdot B$ is short
  - 1^{st} entry: $q_0R < Q\cdot R$
  - i^{th} entry $(i>1)$: $q_0(q_ip+r_i)-q_i(q_0p+r_0)=q_0r_i-q_ir_0$
    - Less than $Q\cdot R$ in absolute value
  - ➔ Total size less than $Q\cdot R\cdot\sqrt{t}$
    - vs. size $\sim Q\cdot P$ (or more) for the basis vectors
- Hopefully we will find it with a lattice-reduction algorithm (LLL or variants)

Will this algorithm succeed?

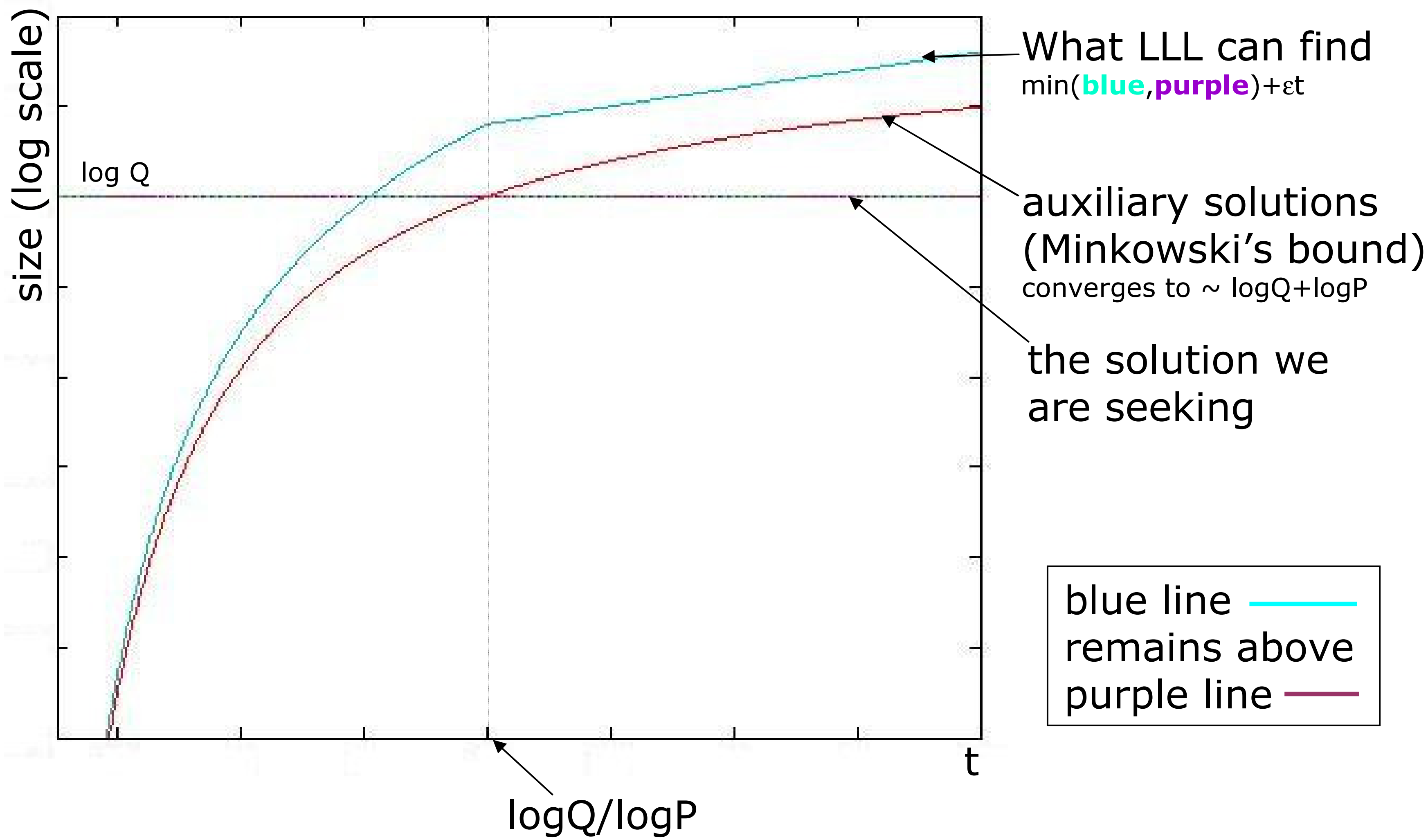
$$\begin{pmatrix} R & x_1 & x_2 & \cdots & x_t \\ & -x_0 & & & \\ & & -x_0 & & \\ & & & \cdots & \\ & & & & -x_0 \end{pmatrix}$$

- Is $\langle q_0, q_1, \ldots, q_t \rangle \cdot B$ shortest in lattice?
  - Is it shorter than $\sqrt{t} \cdot \det(B)^{1/t+1}$? — Minkowski bound
    - det(B) is small-ish (due to R in the corner)
  - Need $((QP)^t R)^{1/t+1} > QR$

    $\Leftrightarrow t+1 > (\log Q + \log P - \log R) / (\log P - \log R)$

    $\sim \log Q / \log P$
- $\log Q = \omega(\log^2 P)$ ➔ need $t = \omega(\log P)$
- Quality of LLL & co. degrades with t
  - Only finds vectors of size $\sim 2^{t/2} \cdot$shortest
    - or $2^{t/2} \rightarrow 2^{\varepsilon t}$ for any constant $\varepsilon > 0$
  - $t = \omega(\log P)$ ➔ $2^{\varepsilon t} \cdot QR > \det(B)^{1/t+1}$
  - Contemporary lattice reduction is not strong enough

Why this algorithm fails



Conclusions

- Fully Homomorphic Encryption is a very powerful tool
- Gentry09 gives first feasibility result
  - Showing that it can be done "in principle"
- We describe a "conceptually simpler" scheme, using only modular arithmetic
- What about efficiency?
  - Computation, ciphertext-expansion are polynomial, but a rather large one…
- Improving efficiency is an open problem

Extra credit

- The hard-core-bit theorem
- Connection between approximate-GCD and simultaneous Diophantine approx.
- Gentry's technique for "squashing" the decryption circuit

Thank you