Homomorphic Encryption
Tutorial

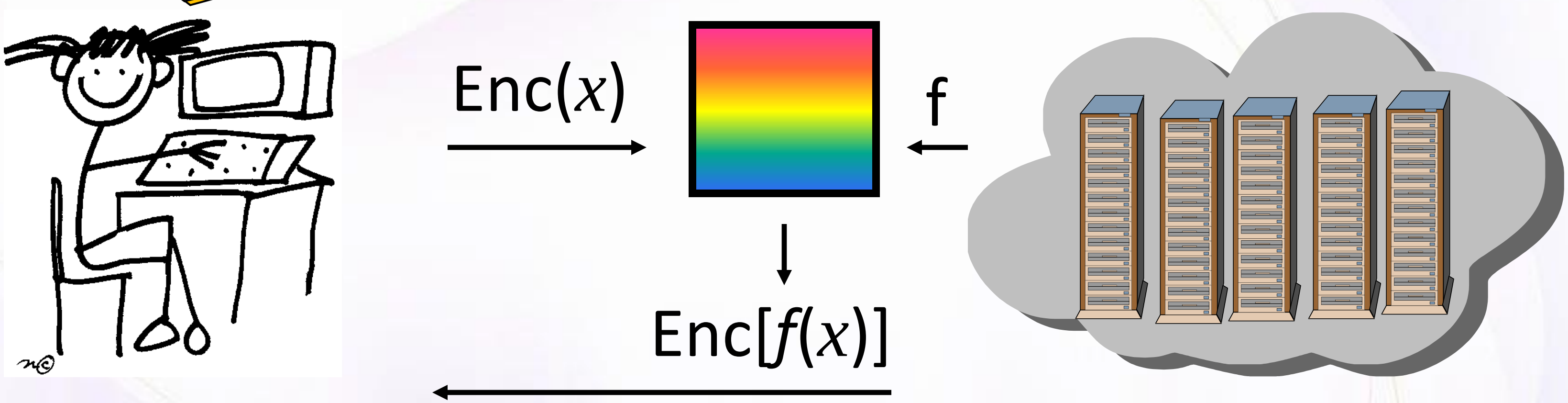
Shai Halevi — IBM
August 2013

Computing on Encrypted Data

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

Outsourcing Computation

"I want to delegate the computation to the cloud, but the cloud shouldn't see my input"

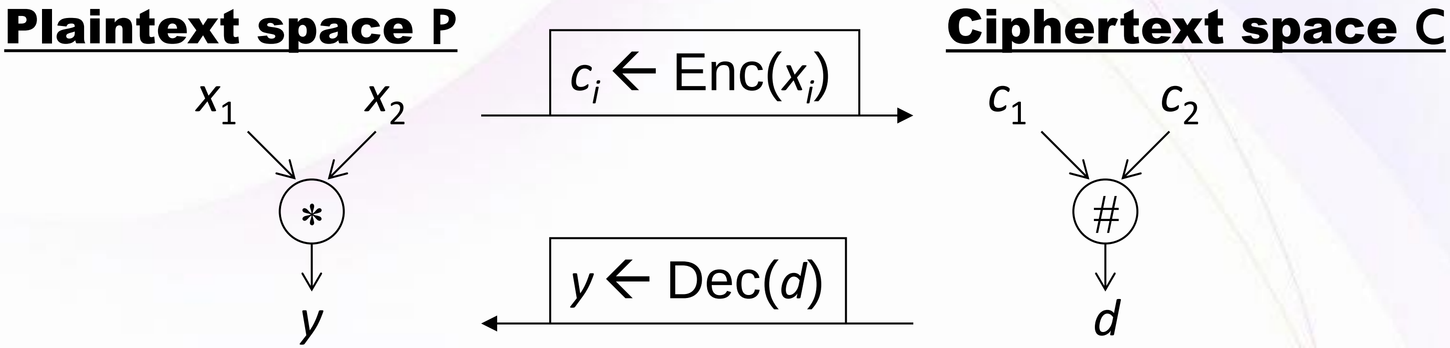


Client
(Input: x)

Server/Cloud
(Function: f)

Privacy Homomorphisms

- Rivest-Adelman-Dertouzos 1978



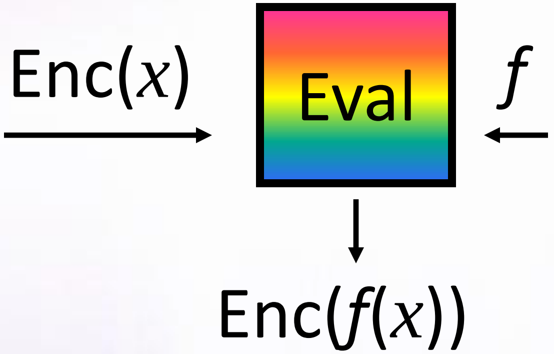
Example: $\mathrm{RSA\_encrypt}_{(e,N)}(x) = x^e \bmod N$

- $x_1^e \times x_2^e = (x_1 \times x_2)^e \bmod N$

"Somewhat Homomorphic": can compute some functions on encrypted data, but not all

"Fully Homomorphic" Encryption

- Encryption for which we can compute arbitrary functions on the encrypted data



Some Notations

- An encryption scheme: (KeyGen, Enc, Dec)
  - Plaintext-space = {0,1}
  - $(pk,sk) \leftarrow$ KeyGen($), $c \leftarrow \text{Enc}_{\text{pk}}(b)$, $b \leftarrow \text{Dec}_{sk}(c)$
- Semantic security [GM'84]:

$$(pk, \text{Enc}_{pk}(0)) \approx (pk, \text{Enc}_{pk}(1))$$

$\approx$ means indistinguishable by efficient algorithms

Homomorphic Encryption (HE)

- H = {KeyGen, Enc, Dec, Eval}

  $c^* \leftarrow \text{Eval}_{pk}(f, \mathbf{c})$

- Homomorphic: $\text{Dec}_{sk}(\underbrace{\text{Eval}_{pk}(f, \text{Enc}_{pk}(x))}_{c^*}) = f(x)$
  - c^* may not look like a "fresh" ciphertext
  - As long as it decrypts to $f(x)$
- Compact: Decrypting c^* easier than computing f
  - Otherwise we could use $\text{Eval}_{pk}(f, \mathbf{c}) = (f, \mathbf{c})$ and $\text{Dec}_{sk}(f, \mathbf{c}) = f(\text{Dec}_{sk}(\mathbf{c}))$
  - Technically, $|c^*|$ independent of the complexity of f

Fully Homomorphic Encryption

- First plausible candidate in [Gen'09]
  - Security from hard problems in ideal lattices
  - Polynomially slower than computing in the clear
    - Big polynomial though
- Many advances since
  - Other hardness assumptions
    - LWE, RLWE, NTRU, approximate-GCD
  - More efficient
  - Other "Advanced properties"
    - Multi-key, Identity-based, …

This Talk

- Regev-like somewhat-homomorphic encryption
  - Adding homomorphism to [Reg'05] cryptosystem
    - Security based on LWE, Ring-LWE
  - Based on [BV'11, BGV'12, B'12]
- Bootstrapping to get FHE [Gen'09]
- Packed ciphertexts for efficiency
  - Based on [SV'11, BGV'12, GHS'12]
- Not in this talk: a new LWE-based scheme
  - [Gentry-Sahai-Waters CRYPTO 2013]

Learning with Errors [Reg'05]

Many equivalent forms, this is one of them:

- Parameters: q (modulus), n (dimension)
- Secret: a random short vector $\boldsymbol{s} \in Z_q^n$
- Input: many pairs $(\boldsymbol{a_i}, b_i)$
  - $\boldsymbol{a}_i \in Z_q^n$ is random, $b_i = \langle \boldsymbol{s}, \boldsymbol{a}_i \rangle + e_i \ (mod\ q)$
    - e_i is short
- Goal: find the secret $\boldsymbol{s}$
  - Or distinguish $(\boldsymbol{a}_i, b_i)$ from random in Z_q^{n+1}

[Regev'05, Peikert'09]: As hard as some worst-case lattice problems in dim n (for certain range of params)

Regev's Cryptosystem [Reg'05]

- The shared-key variant (enough for us)
- Secret key: vector $\boldsymbol{s}'$, denote $\boldsymbol{s} = (\boldsymbol{s}', \mathbf{1})$
- Encrypt($\sigma \in \{0,1\}$)
  - $\boldsymbol{c} = (\boldsymbol{a}, b)$ s.t. $b = \sigma \frac{q}{2} - \langle \boldsymbol{s}', \boldsymbol{a} \rangle + e \ (mod\ q)$
  - Convenient to write $\langle \boldsymbol{s}, \boldsymbol{c} \rangle = \sigma \frac{q}{2} + e \ (mod\ q)$
- Decrypt($\boldsymbol{s}, \boldsymbol{c}$)
  - Output 0 if $|\langle \boldsymbol{s}, \boldsymbol{c} \rangle$ mod q$| \leq q/4$, else output 1
  - Correct decryption as long as error $< q/4$

Security: If LWE is hard, cipehrtext is pseudorandom

Additive Homomorphism

- If $\langle \boldsymbol{s}, \boldsymbol{c}_i \rangle \approx \sigma_i \frac{q}{2}$ (mod q) then
  $$\langle \boldsymbol{s}, \boldsymbol{c}_{\mathbf{1}} + \boldsymbol{c}_2 \rangle \approx (\sigma_1 \oplus \sigma_2) \frac{q}{2} \text{ (mod q)}$$
- Error doubles on addition
- Correct decryption as long as the error $< q/4$

How to Multiply [BV'11, B'12]

- Step 1: Tensor Product
  - If $\langle \boldsymbol{s}, \boldsymbol{c}_i \rangle \approx \sigma_i \frac{q}{2}$ (mod q) and **s** is small ($|\boldsymbol{s}| \ll q$) then $\langle \boldsymbol{s} \otimes \boldsymbol{s}, \boldsymbol{c}_1 \otimes \boldsymbol{c}_2 \rangle \approx \sigma_1 \sigma_2 \frac{q^2}{4}$ (mod q^2)
    - Error has extra additive terms of size $\approx |s| \cdot q \ll q^2$
  - So $\boldsymbol{c}^* = round((\boldsymbol{c}_1 \otimes \boldsymbol{c}_2)/\frac{q}{2})$ encrypts $\sigma_1 \sigma_2$ relative to secret key $\boldsymbol{s}^* = (\boldsymbol{s} \otimes \boldsymbol{s})$
    - Rounding adds another small additive error
  - But the dimension squares on multiply

How to Multiply [BV'11, B'12]

- Step 2: Dimension Reduction
  - Publish "key-switching gadget" to ranslate $\boldsymbol{c}^*$ wrt $\boldsymbol{s}^*$ ➔ $\boldsymbol{c}$ wrt $\boldsymbol{s}$
    - Essentially an encryption of $\boldsymbol{s}^*$ under $\boldsymbol{s}$
  - $n \times n^2$ rational matrix W s.t. $\boldsymbol{s}^{\boldsymbol{T}} \times W \approx \boldsymbol{s}^* (mod\ q)$
  - Given $\boldsymbol{c}^*$, compute $\mathbf{c} \leftarrow \mathrm{Round}(W \times \boldsymbol{c}^*)\ (mod\ q)$
  - $\langle \boldsymbol{s}, \boldsymbol{c} \rangle \approx \boldsymbol{s}^{\boldsymbol{T}} \times W \times \boldsymbol{c}^* \approx \langle \boldsymbol{s}^*, \boldsymbol{c}^* \rangle \approx \sigma \frac{q}{2}\ (mod\ q)$
    - Some extra work to keep error from growing too much
    - Still secure under reasonable hardness assumptions

Somewhat Homomorphic Encryption

- Error doubles on addition, grows by poly(n) factor on multiplication (e.g., n^2 factor)
  - When computing a depth-d circuit we have $|\text{output-error}| \leq |\text{input-error}| \cdot n^{2d}$
- Setting parameters:
  - Start from $|\text{input-error}| \leq n^d$ (say)
  - Set $q > 4n^d \cdot n^{2d} = 4n^{3d}$
  - Set the dimension large enough to get security
- $|\text{output-error}| < q/4$, so no decryption errors

FHE via Bootstrapping [Gen'09]

- So far, circuits of pre-determined depth

x_1

x_2

...

x_t

$C(x_1, x_2, \ldots, x_t)$

FHE via Bootstrapping [Gen'09]

- So far, circuits of pre-determined depth

x_1
x_2
...
x_t

C

$C(x_1, x_2, \ldots, x_t)$

- Can eval $y=C(x_1,x_2\ldots,x_n)$ when x_i's are "fresh"
- But y is an "evaluated ciphertext"
  - Can still be decrypted
  - But eval $C'(y)$ will increase noise too much

FHE via Bootstrapping [Gen'09]

- So far, circuits of pre-determined depth

- Bootstrapping to handle deeper circuits
  - We have a noisy evaluated ciphertext y
  - Want to get another y with less noise

FHE via Bootstrapping [Gen'09]

- For ciphertext c, consider $\mathbf{D}_c(sk) = \mathrm{Dec}_{sk}(c)$
  - Hope: $\mathrm{D}_c(*)$ is a low-depth circuit (on input sk)
- Include in the public key also $\mathrm{Enc}_{pk}(sk)$

- Homomorphic computation applied only to the "fresh" encryption of sk

FHE via Bootstrapping [Gen'09]

- Similarly define $\mathbf{M}_{c_1,c_2}(sk)$ = $\text{Dec}_{sk}(c_1)\cdot\text{Dec}_{sk}(c_1)$

y_1 y_2

c_1 c_2

sk_1

sk_2

...

sk_n

M_{c_1,c_2}

c'

$\text{M}_{c_1,c_2}(sk)$

= $\text{Dec}_{sk}(c_1)$ x $\text{Dec}_{sk}(c_2)$ = y_1 x y_2

- Homomorphic computation applied only to the "fresh" encryption of sk

(In)Efficiency of This Scheme

- The LWE-based somewhat-homomorphic scheme has depth-$\tilde{O}(\log qn)$ decryption circuit
- To get FHE need modulus $q \geq 2^{polylog(k)}$ and dimension $\mathrm{n} \geq \tilde{\Omega}(k)$
  - k is the security parameter
- The ciphertext-size is $\tilde{\Omega}(k)$ bits
- Key-switching matrix is of size $\tilde{\Omega}(k^3)$ bits
  - ➔ Each multiplication takes at least $\tilde{\Omega}(k^3)$ times
  - ➔ $\tilde{\Omega}(k^3)$ slowdown vs. computing in the clear

Better Efficiency with Ring-LWE

- Replace Z by Z[X]/F(X)
  - F is a degree-d polynomial with $d = \tilde{\Theta}(k)$
- Can get security with lower dimension
  - $n = \tilde{\Theta}(k/d)$, as low as $n = 2$
- The ciphertext-size still $\tilde{\Omega}(k)$ bits
- But key-switching matrix size only $\tilde{\Theta}(k)$ bits
  - It includes $n^2 \times n = 8$ ring elements

➔ $\tilde{\Theta}(k)$ slowdown vs. computing in the clear

Ciphertext Packing

- Cannot reduce ciphertext size below $\widetilde{\Theta}(k)$
- But we can pack more bits in each ciphertext
- Recall decryption: $ptxt \leftarrow MSB(\langle \mathbf{s}, \mathbf{c} \rangle \ mod\ q)$
  - $ptxt$ is a polynomial in $\mathrm{R}_2 = Z[X]/(F(X), 2)$
- Use cyclotomic rings, $F(X) = \Phi_m(X)$
- Use CRT in R_2 to pack many bits inside m
  - The cryptosystem remains unchanged
  - Encoding/decoding of bits inside plaintext polys

Plaintext Algebra

- $\Phi_m(X)$ irreducible over Z, but not mod 2
  - $\Phi_m(X) \equiv \prod_{j=1}^{\ell} F_j(X)$ (mod 2)
  - F_j's are irreducible, all have the same degree d
    - degree d is the order of 2 in Z_m^*
  - For some m's we can get $\ell = \frac{\phi(m)}{d} = \Omega(\frac{\text{m}}{\text{log m}})$
- $\mathrm{R}_2 = Z_2[X]/\Phi_m$ is a direct sum, $\mathrm{R}_2 = \bigoplus_j R_{2,j}$
  - $R_{2,j} = Z_2[X]/F_j(X) \cong GF(2^d)$
- 1-1 mapping $a \in R_2 \leftrightarrow [\alpha_1, \dots, \alpha_\ell] \in GF\left(2^d\right)^\ell$

Plaintext Slots

- Plaintext $a \in R_2$ encodes ℓ values $\alpha_j \in GF(2^d)$
  - To embed plaintext bits, use $\alpha_j \in GF(2) \subset GF(2^d)$
- Ops +,× in R_2 work independently on the slots
  - ℓ-ADD: $a + a' \cong [\alpha_1 + \alpha_1', \ldots, \alpha_\ell + \alpha_\ell']$
  - ℓ-MUL: $a \times a' \cong [\alpha_1 \times \alpha_1', \ldots, \alpha_\ell \times \alpha_\ell']$
- If $\ell = \widetilde{\Omega}(k)$ then our $\widetilde{\Theta}(k)$-bit ciphertext can hold $\widetilde{\Omega}(k)$ plaintext bits
  - Ciphertext-expansion ratio only polylog(*k*)

Aside: an ℓ-SELECT Operation

x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	0	1	0	1	0

x =

x_1	0	0	x_4	0	x_6	0

+

x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}
0	1	1	0	1	0	1

x =

0	x_9	x_{10}	0	x_{12}	0	x_{14}

---

x_1	x_9	x_{10}	x_4	x_{12}	x_6	x_{14}

- We will use this later

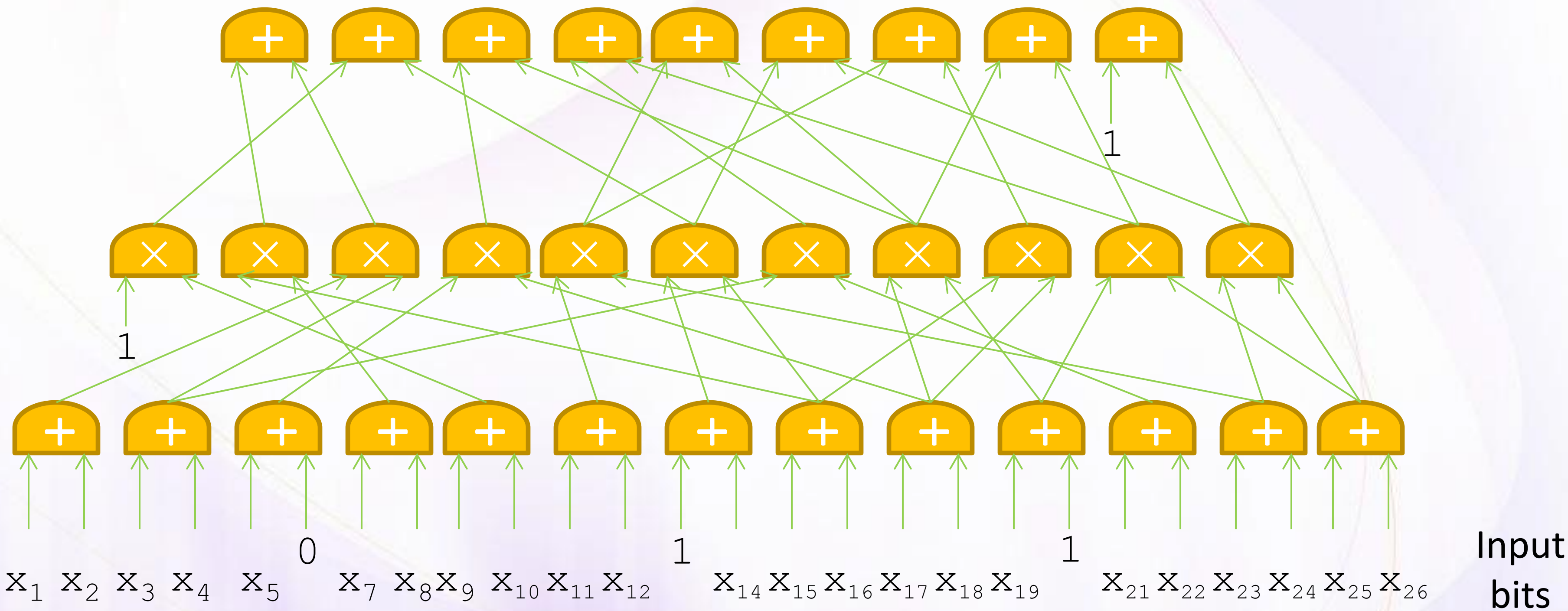
Homomorphic SIMD [SV'11]

- SIMD = Single Instruction Multiple Data
- Computing the same function on ℓ inputs at the price of one computation
  - Overhead only polylog(k)
- Pack the inputs into the slots
  - Bit-slice, inputs to j'th instance go in j'th slots
- Compute the function once
- After decryption, decode the ℓ output bits from the output plaintext polynomial

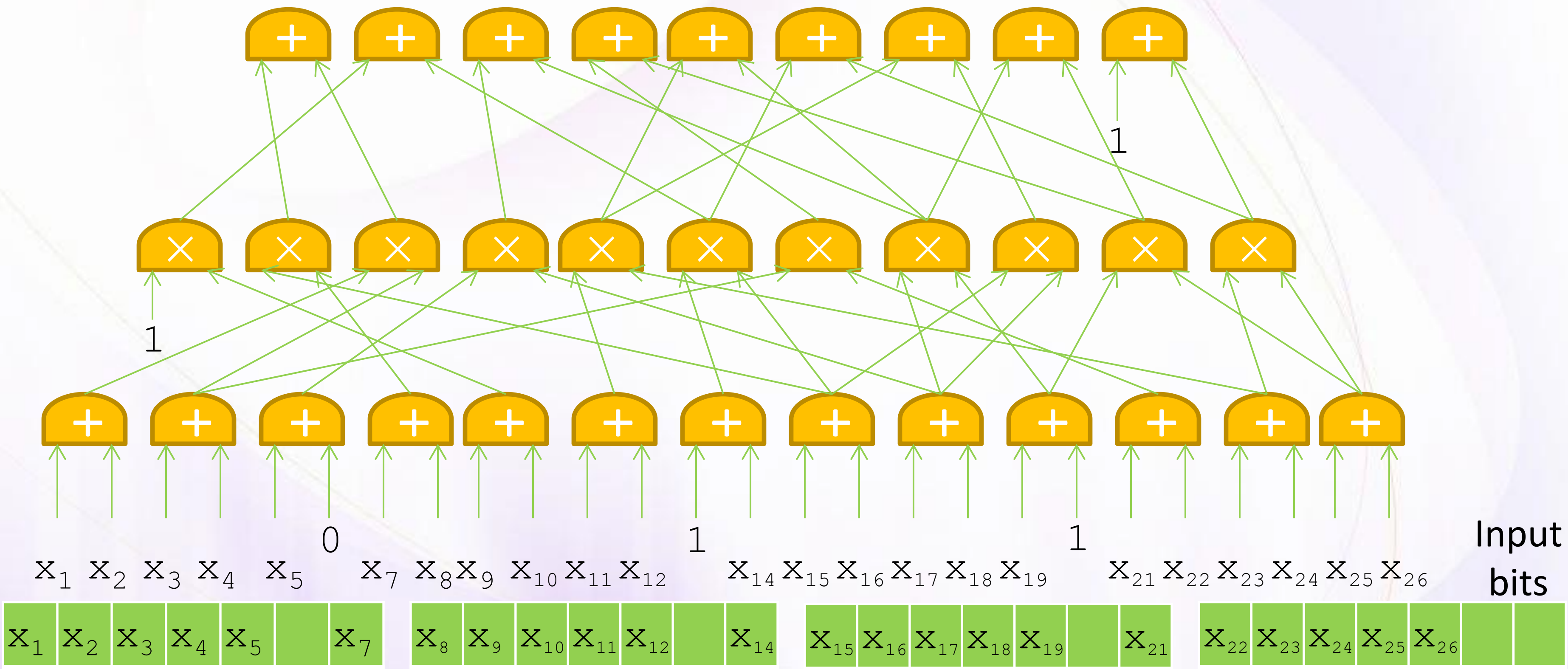
Beyond SIMD Computation

- To reduce overhead for a single computation:
  - Pack all input bits in just a few ciphertexts
  - Compute while keeping everything packed
- How to do this?

So you want to compute some function…



So you want to compute some function using SIMD...



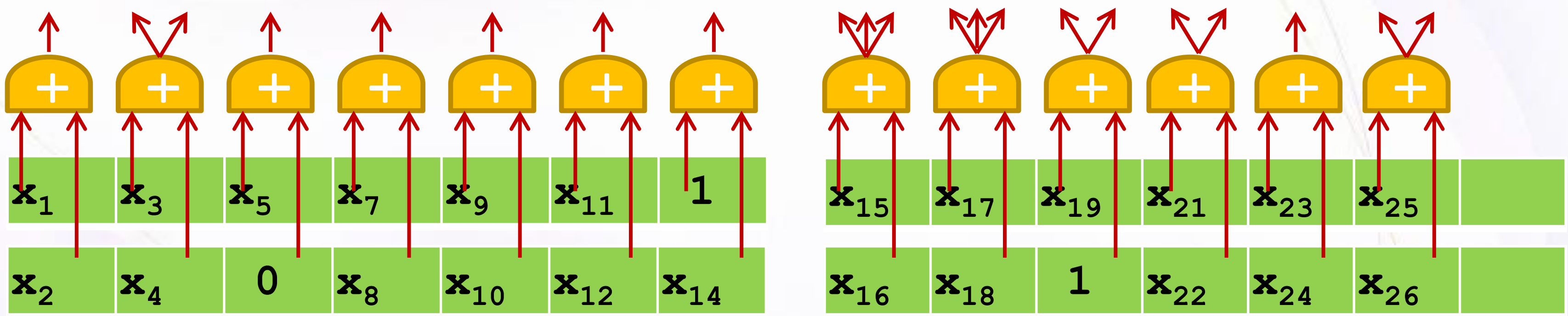
Routing Values Between Levels

- We need to map this

x_1	x_2	x_3	x_4	x_5	0	x_7
x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	1	x_{21}

x_8	x_9	x_{10}	x_{11}	x_{12}	1	x_{14}
x_{22}	x_{23}	x_{24}	x_{25}	x_{26}		

- Into that … so we can use ℓ-add



- Is there a natural operation on polynomials that moves values between slots?

Using Automorphisms

- The operation $\kappa_t: a(X) \mapsto a(X^t) \in R_2$
- Under some conditions on m, exists $t \in Z_m^*$ s.t.,
  - For any $a \in R_2$ encoding $a \leftrightarrow [\alpha_1, \alpha_2, \ldots, \alpha_\ell]$,
    $$\kappa_t(a) \leftrightarrow [\alpha_2, \ldots, \alpha_\ell, \alpha_1]$$
  - t is a generator of $Z_m^*/(2)$ (if it exists)
- Once we have rotations, we can get every permutation on the plaintext slots
  - Using only $O(\log \ell)$ shifts and SELECTs [GHS'12]
- How to implement κ_t homomorphically?

Homomorphic Automorphism

- Recall decryption via inner product $\langle \boldsymbol{s}, \boldsymbol{c} \rangle \in R_q$
  - If $a(X) = \langle \boldsymbol{s}(X), \boldsymbol{c}(X) \rangle \; mod \; (\Phi_m(X), q)$ then also $a(X^t) = \langle \boldsymbol{s}(X^t), \boldsymbol{c}(X^t) \rangle \; mod \; (\Phi_m(X^t), q)$
  - Since $\Phi_m(X) | \Phi_m(X^t)$ for any $t \in Z_m^*$, then also $a(X^t) = \langle \boldsymbol{s}(X^t), \boldsymbol{c}(X^t) \rangle \; mod \; (\Phi_m(X), q)$
- Therefore $\boldsymbol{c}' = \kappa_t(\boldsymbol{c})$ is an encryption of $a' = \kappa_t(a)$ relative to key $\boldsymbol{s}' = \kappa_t(\boldsymbol{s})$
- Can publish key-switching matrix $W[\boldsymbol{s}' \to \boldsymbol{s}]$ to get back an encryption relative to $\boldsymbol{s}$

Summary of RLWE HE encryption

- Native plaintext space $\mathrm{R}_2 = Z_2[X]/\Phi_m$
  - $a \in R_2$ used to pack ℓ values $\alpha_j \in GF(2^d)$
- sk is $s \in R_q$, ctxt is a pair $(c_0, c_1) \in R_q^2$
- Decryption is $a := MSB(\langle (c_0, c_1), (s, 1) \rangle)$
  - Inner product over R_q
- Homomorphic addition, multiplication work element-size on the α_j's
- Homomorphic automorphism to move α_j's between the slots