Winter School on Lattice-Based Cryptography and Applications
Bar-Ilan University, Israel 19/2/2012-22/2/2012

Bar-Ilan University
Dept. of Computer Science

Fully Homomorphic Encryption

Craig Gentry
IBM Watson

Outline for Today

Bar-Ilan University
Dept. of Computer Science

- Optimizations of Somewhat Homomorphic Encryption (SWHE)
- Constructions of Fully Homomorphic Encryption (FHE)

Bar-Ilan University
Dept. of Computer Science

SWHE for Bounded *Depth* Circuits [BGV12]

And Better Management of Ciphertext Noise…

Bar-Ilan University
Dept. of Computer Science

Review of [BV11b]: SWHE Based on LWE

- Focusing on the “noise problem”…

The LWE Problem

Bar-Ilan University
Dept. of Computer Science

- Noisy Polly Cracker Version:
  - Let χ be an error distribution.
  - Distinguish these distributions:
    - Generate uniform $s \leftarrow Z_q^n$. For many i, generate $e_i \leftarrow \chi$ and a linear polynomial $f_i(x_1, \ldots, x_n) = f_0 + f_1 x_1 + \ldots + f_n x_n$ (from Z_q^{n+1}) such that $[f_i(s_1, \ldots, s_n)]_q = e_i$.
    - For many i, generate and output a uniformly random linear polynomial $f_i(x_1, \ldots, x_n)$ (from Z_q^{n+1}).

SWHE Based on LWE [BV11b]

Bar-Ilan University
Dept. of Computer Science

- Parameters: q such that gcd(q,2)=1.
- KeyGen: Secret = uniform $\mathbf{s} \in Z_q^n$. Public key: linear polys $\{f_i(x_1,\ldots,x_n)\}$ s.t. $[f_i(\mathbf{s})]_q = 2e_i$, $|e_i| \ll q$.
- Encrypt: Set $g(x_1,\ldots,x_n)$ as a random subset sum of $\{f_i(x_1,\ldots,x_n)\}$. Output $c(x_1,\ldots,x_n) = m + g(x_1,\ldots,x_n)$.
- Decrypt: $[c(\mathbf{s})]_q = m + \text{smeven}$. Reduce mod 2.
- ADD and MULT:
  - Output sum or product of ciphertext polynomials.
- Relinearize / Key-Switch

The Noise Problem

Bar-Ilan University
Dept. of Computer Science

- ADD: $c(\mathbf{x}) = c_1(\mathbf{x}) + c_2(\mathbf{x})$.
  - Noise of $c(\mathbf{x})$ – namely, $[c(\mathbf{s})]_q$ – is sum of noises.
- MULT: $c(\mathbf{x}) = c_1(\mathbf{x}) \cdot c_2(\mathbf{x})$.
  - Noise $[c(\mathbf{s})]_q$ is product of noises.
  - Sort of… After MULT, there is "relinearization" step that adds a small amount to the noise.
- Function F: $c(\mathbf{x}) \approx F(c_1(\mathbf{x}),\ldots,c_t(\mathbf{x}))$.
  - Noise $[c(\mathbf{s})]_q \approx f(c_1(\mathbf{s}),\ldots,c_t(\mathbf{s}))$ – i.e., F applied to noises.
  - Rough approximation:
    - If F has degree d and fresh noises are bounded by B, $c(\mathbf{x})$ has noise B^d.
    - Noise magnitude increases exponentially with degree.

The Noise Problem Hurts Efficiency. Why?

Bar-Ilan University
Dept. of Computer Science

- SWHE ciphertexts must be large to let noise "room to grow".
- "Noise" grows exponentially with degree. To successfully evaluate degree-d poly, noise $B \rightsquigarrow B^d$ without "wrapping".
- So, coefficients of lattice vectors have $> d$ bits.
- For security, we need it to be hard to $B^{d-1} > 2^d$-approximate lattice problems in 2^k time.
  - Requires lattice dim $> d \cdot k$.
  - Total ciphertext length $> d^2 \cdot k$ bits.

SWHE for Bounded *Degree*

Bar-Ilan University
Dept. of Computer Science

- Since total ciphertext length $\approx d^2 \cdot k$ bits, we have SWHE for bounded degree:
- SWHE for bounded degree: A family of schemes $E^{(d)}$, $d \in Z$, that for security parameter k,
  - $E^{(d)}$ can homomorphically evaluate functions of degree d.
  - KeyGen, Enc, Dec, ADD, MULT are all poly(k,d).
  - Eval has complexity polynomial in k, d, and circuit size.

This is the best we can hope for when noise grows exponentially with degree.

Bar-Ilan University
Dept. of Computer Science

SWHE for Bounded *Depth…*

SWHE for Bounded Depth Circuits

Bar-Ilan University
Dept. of Computer Science

- "Leveled FHE" [Gen09]: Relaxation of FHE… A family of schemes $E^{(L)}$, $L \in Z$, is "leveled fully homomorphic" if, for security parameter k,
  - $E^{(L)}$ can homomorphically evaluate circuits of depth L,
  - The Dec (decrypt) function is the same for all L,
  - KeyGen, Enc, Dec, ADD, MULT are all poly(k,L).
  - Eval has complexity polyomial in k, L, and circuit size.
- Humbler name for it: "SWHE for bounded *depth* circuits".

Better Noise Management?

Bar-Ilan University
Dept. of Computer Science

- Our fantasy:
  - Noise doesn't grow exponentially with degree.
  - There is some *simple* trick to reduce noise after MULTs.
  - We get better noise management, hence shorter ciphertexts and SWHE for bounded depth.

Better Noise Management?

Bar-Ilan University
Dept. of Computer Science

- Crazy Idea [BV11b, BGV12]:
  - Suppose c encrypts m – that is, $m = [[c(\mathbf{s})]_q]_2$.
  - Let's pick $p<q$ and set $c^*(\mathbf{x}) = (p/q) \cdot c(\mathbf{x})$, rounded.
  - Maybe it is true that:
    - $c^*(\mathbf{x})$ encrypts m: $m = [[c^*(\mathbf{s})]_p]_2$ (new inner modulus).
    - $|[c^*(\mathbf{s})]_p| \approx (p/q) \cdot |[c(\mathbf{s})]_q|$ (noise is smaller).
  - This really shouldn't work…

Modulus Reduction Trick

Bar-Ilan University
Dept. of Computer Science

- Scaling lemma: Let $p < q$ be odd moduli.
  - Given $\mathbf{c}$ with $m = [[<\mathbf{c},\mathbf{s}>]_q]_2$. Set $\mathbf{c'} = (p/q)\mathbf{c}$. Set $\mathbf{c''}$ to be
    - the integer vector closest to $\mathbf{c'}$
    - such that $\mathbf{c''} = \mathbf{c} \bmod 2$.
  - If $|[<\mathbf{c},\mathbf{s}>]_q| < q/2 - (q/p)\cdot \ell_1(\mathbf{s})$, then $\mathbf{c''}$ is a valid encryption of m with possibly much less noise!
    - $m = [<\mathbf{c''},\mathbf{s}>]_p]_2$.
    - $|[<\mathbf{c''},\mathbf{s}>]_p| < (p/q) \cdot |[<\mathbf{c},\mathbf{s}>]_q| + \ell_1(\mathbf{s})$, where $\ell_1(\mathbf{s})$ is ℓ_1-norm of $\mathbf{s}$.

Modulus Reduction Trick

Bar-Ilan University
Dept. of Computer Science

Annotated Proof

1. For some k, $[\langle \mathbf{c},\mathbf{s}\rangle]_q = \langle \mathbf{c},\mathbf{s}\rangle - kq$.
2. $(p/q)[\langle \mathbf{c},\mathbf{s}\rangle]_q = \langle \mathbf{c'},\mathbf{s}\rangle - kp$.
3. $|\langle \mathbf{c''}-\mathbf{c'},\mathbf{s}\rangle| < \ell_1(\mathbf{s})$.

1. Imagine $\langle \mathbf{c},\mathbf{s}\rangle$ is close to kq.
2. Then $\langle \mathbf{c'},\mathbf{s}\rangle$ is close to kp.
3. $\langle \mathbf{c''},\mathbf{s}\rangle$ close to kp if **s** is small.

4. Thus, $|\langle \mathbf{c''},\mathbf{s}\rangle - kp| < (p/q)\,|[\langle \mathbf{c},\mathbf{s}\rangle]_q| + \ell_1(\mathbf{s}) < p/2$.
5. So, $[\langle \mathbf{c''},\mathbf{s}\rangle]_p = \langle \mathbf{c''},\mathbf{s}\rangle - kp$.
6. Since $\mathbf{c'} = \mathbf{c}$ and $p = q \bmod 2$, we have $[\langle \mathbf{c''},\mathbf{s}\rangle]_p]_2 = [\langle \mathbf{c},\mathbf{s}\rangle]_q]_2$.

Scaling lemma: Let $p<q$ be odd moduli.

- Given **c** with $m = [[\langle \mathbf{c},\mathbf{s}\rangle]_q]_2$. Set $\mathbf{c'} = (p/q)\mathbf{c}$. Set $\mathbf{c''}$ to be
  - the integer (ring) vector closest to $\mathbf{c'}$ such that $\mathbf{c''} = \mathbf{c} \bmod 2$.
- If $|[\langle \mathbf{c},\mathbf{s}\rangle]_q| < q/2 - (q/p)\cdot \ell_1(\mathbf{s})$, then:
  - $\mathbf{c''}$ is a valid encryption of m with possibly much less noise!
  - $m = [\langle \mathbf{c''},\mathbf{s}\rangle]_p]_2$, and $|[\langle \mathbf{c''},\mathbf{s}\rangle]_p| < (p/q)\cdot |[\langle \mathbf{c},\mathbf{s}\rangle]_q| + \ell_1(\mathbf{s})$.

Modulus Reduction Example

Bar-Ilan University
Dept. of Computer Science

- Example: q=127, p=29, **c**=(175,212), **s**=(2,3)
- <**c**,**s**> mod q = 986−8·127 = −30
- **c'** = (p/q) · **c** = (39.9,48.4)
  - To get **c"**, we round down both values (39,48).
- <**c"**,**s**> mod p = 222−8·29 = −10
- k=8 in both cases, and −30=−10 mod 2.
- The noise magnitude decreases from 30 to 10.
  - But relative magnitude increases: 10/29 > 30/127.

Small Secret Key

Bar-Ilan University
Dept. of Computer Science

- Recall $|[<\mathbf{c}',\mathbf{s}>]_p| < (p/q) \cdot |[<\mathbf{c},\mathbf{s}>]_q| + \ell_1(\mathbf{s})$.
- Luckily [ACPS 2009] proved that LWE is hard even when **s** is small
  - chosen from the error distribution χ.
  - So we use this distribution for the secret keys.

Modulus Reduction in RLWE-Based SWHE

Bar-Ilan University
Dept. of Computer Science

- Scaling lemma also holds for LPR10, BV11a.
- [LPR10]: Ring-LWE encryption scheme can can also have small secret keys, from the error distribution χ.

How Does Modulus Switching Help?

Bar-Ilan University
Dept. of Computer Science

To evaluate a circuit of depth L…

- Start with a large modulus q_L and noise $\eta \ll q_L$.
- After first MULT, noise grows to η^2.
- Switch the modulus to $q_{L-1} \approx q_L/\eta$.
  - Noise reduced to $\eta^2/\eta \approx \eta$.
- After next MULT, noise again grows to η^2. Switch to $q_{L-2} \approx q_{L-1}/\eta$ to reduce the noise to η.
- Keep switching moduli after each layer.
  - Setting $q_{i-1} \approx q_i/\eta$. ("Ladder" of decreasing moduli.)
  - Until the last modulus just barely satisfies $q_1 > \eta$.

How Does Modulus Switching Help?

Bar-Ilan University
Dept. of Computer Science

- Example: $q_9 \approx n^9$ with modulus reduction.

	Noise	Modulus
Fresh ciphertexts	η	$q_9 = \eta^9$
Level-1, Degree=2	η	$q_8 = \eta^8$
Level-2, Degree=4	η	$q_7 = \eta^7$
Level-3, Degree=8	η	$q_6 = \eta^6$
Level-4, Degree=16	η	$q_5 = \eta^5$
Level-5, Degree=32	η	$q_4 = \eta^4$
Level-6, Degree=64	η	$q_3 = \eta^3$
Level-7, Degree=128	η	$q_2 = \eta^2$
Level-8, Degree=256	η	$q_1 = \eta$

How Does Modulus Switching Help?

Bar-Ilan University
Dept. of Computer Science

- Example: $q_9 \approx n^9$ with no modulus reduction.

	Noise	Modulus
Fresh ciphertexts	η	$q_9 = \eta^9$
Level-1, Degree=2	η^2	$q_9 = \eta^9$
Level-2, Degree=4	η^4	$q_9 = \eta^9$
Level-3, Degree=8	η^8	$q_9 = \eta^9$
Level-4, Degree=16	η^{16}	$q_9 = \eta^9$
Level-5, Degree=32	η^{32}	
Level-6, Degree=64	η^{64}	
Level-7, Degree=128	η^{128}	
Level-8, Degree=256	η^{256}	

Decryption error

Performance

Bar-Ilan University
Dept. of Computer Science

- To evaluate circuit of depth L;
  - Largest modulus is $q_L \approx q_1^L \approx \eta^L$.
  - Largest ciphertext is $O(k \cdot \text{poly}(L))$ bits, where k is the security parameter.
- Compare: without modulus reduction:
  - ciphertext was $O(k \cdot d^2)$ bits, where d was the *degree* (not the *depth*) of the circuit.
- *Depth* is logarithmic in *degree*.
- Exponential improvement.

The Final Ciphertext

Bar-Ilan University
Dept. of Computer Science

- Final ciphertext (at output level) is small
  - q_1 is small.
  - Use key-switching to reduce dimension of the ciphertext if needed ("dimension reduction" [BV11b]).
  - Final ciphertext can be as small as a normal (non-homomorphic) Regev'05 ciphertext.
- We have SWHE for bounded depth circuits.

Security

Bar-Ilan University
Dept. of Computer Science

- Based on (R)LWE, but for what approx factor?
- Approx factor = modulus/|noise| = $(\text{poly}(k))^{\text{depth}}$.
  - Previously, modulus/|noise| = $(\text{poly}(k))^{\text{degree}}$.

SWHE over the Integers for Bounded Depth Circuits

Bar-Ilan University
Dept. of Computer Science

- [CNT12] extends the modulus reduction trick to the integer scheme.

Bar-Ilan University
Dept. of Computer Science

Batched Computation on Encrypted Data [SV11]

Each ciphertext is “packed” with an array of plaintexts…

The Setting

Bar-Ilan University
Dept. of Computer Science

- Ciphertexts are long, plaintexts are often short.
- Wasteful!
- Overhead of homomorphic encryption
  - = (encrypted comp. time)/(unencrypted comp. time)
  - > (ciphertext length)/(plaintext length)

Batching / Packing Ciphertexts

Bar-Ilan University
Dept. of Computer Science

- Each ciphertext has an *array* of "plaintext slots".
- An operation (+,x) on a ciphertext acts separately, in parallel, on each "plaintext slot" (each index in array).
  - Suppose two ciphertexts c and c' have (b_1,b_2,b_3) and (b_1',b_2',b_3') respectively in their "slots"
  - 3-ADD(c,c') $\rightarrow (b_1+b_1',\ b_2+b_2',\ b_3+b_3')$.
  - 3-MULT(c,c') $\rightarrow (b_1 \cdot b_1',\ b_2 \cdot b_2',\ b_3 \cdot b_3')$.
  - 3-ADD, 3-MULT cost same as ADD, MULT.
- Think Chinese Remainder Theorem.

LWE-Based SWHE [BV11b]

Bar-Ilan University
Dept. of Computer Science

- Parameters: q such that gcd(q,2)=1.
- KeyGen: Secret = uniform $\mathbf{s} \in Z_q^n$. Public key: linear polys $\{f_i(x_1,\ldots,x_n)\}$ s.t. $[f_i(\mathbf{s})]_q = 2e_i$, $|e_i| \ll q$.
- Encrypt ($m \in Z_2$): Set $g(x_1,\ldots,x_n)$ as a random subset sum of $\{f_i(x_1,\ldots,x_n)\}$. Output $c(x_1,\ldots,x_n)=m+g(x_1,\ldots,x_n)$.
- Decrypt: $[c(\mathbf{s})]_q = m+\text{smeven}$. Reduce mod 2.

- ADD and MULT:
  - Output sum or product of ciphertext polynomials.

LWE-Based SWHE [BV11b]

Bar-Ilan University
Dept. of Computer Science

- Parameters: q and small p_1,p_2,p_3 s.t. $\gcd(q,p_1p_2p_3)=1$.
- KeyGen: Secret = uniform $\mathbf{s} \in Z_q^n$. Public key: linear polys $\{f_i(x_1,\ldots,x_n)\}$ s.t. $[f_i(\mathbf{s})]_q=p_1p_2p_3e_i$, $|e_i| \ll q$.
- Encrypt ($m \in Z_{p1p2p3}$): Set $g(x_1,\ldots,x_n)$ as a random subset sum of $\{f_i(x_1,\ldots,x_n)\}$. Output $c(x_1,\ldots,x_n)=m+g(x_1,\ldots,x_n)$.
- Decrypt: $[c(\mathbf{s})]_q = m+$(mult of $p_1p_2p_3$). Reduce mod $p_1p_2p_3$.

- ADD and MULT:
  - Output sum or product of ciphertext polynomials.
  - By CRT, ADD and MULT operate separately on $\{m \bmod p_i\}$.

Batching in RLWE-Based SWHE

Bar-Ilan University
Dept. of Computer Science

- Motivation: Better efficiency:
  - RLWE more efficient than LWE even in non-batched setting.
  - Batching works very well in RLWE-based SWHE.

CRT over Polynomial Rings

Bar-Ilan University
Dept. of Computer Science

- Let $R = Z[y]/h(y)$, p prime, $R_p = Z_p[y]/h(y)$.
- Suppose $h(y) = \prod h_i(y) \bmod p$.
- Then $R_p \equiv$ Direct product of $\{Z_p[y]/h_i(y)\}$.
- Example:
  - $R = Z[y]/(y^4+1)$, $p=17$.
  - $(y^4+1) = (y-2)(y-8)(y-15)(y-9) \bmod 17$
    - $2, 8=2^3, 15=2^5, 9=2^7$ are the primitive 8-th roots of unity mod 17.
  - $Z_{17}[y]/(y^4+1) \equiv$ Direct product of $Z_{17}[y]/(y-2)$, $Z_{17}[y]/(y-8)$, …
  - $m(y) \in Z_{17}[y]/(y^4+1)$ is determined by its evaluations at 2, 8, 15, 9.

Recall SWHE from RLWE [BV11a]

Bar-Ilan University
Dept. of Computer Science

- Parameters: q with gcd(q,2)=1, $R = Z[y]/(y^n+1)$, $R_2 = Z_2[y]/(y^n+1)$, $R_q = Z_q[y]/(y^n+1)$.
- KeyGen: Secret = uniform $s \in R$. Public key: linear polys $\{f_i(x)\}$ s.t. $f_i(s)=2e_i$, $|e_i| \ll q$.
- Encrypt($m \in R_2$): : Set g(x) as a random subset sum of $\{f_i(x)\}$. Output $c(x)=m+g(x)$.
- Decrypt: $c(s) = m+2e$. Reduce mod 2.

- ADD and MULT: Add or multiply the ciphertext polynomials.

Recall SWHE from RLWE [BV11a]

Bar-Ilan University
Dept. of Computer Science

- Parameters: p, q with gcd(q,p)=1, $R = Z[y]/(y^n+1)$, $R_p = Z_p[y]/(y^n+1)$, $R_q = Z_q[y]/(y^n+1)$.
- KeyGen: Secret = uniform $s \in R$. Public key: linear polys $\{f_i(x)\}$ s.t. $f_i(s) = pe_i$, $|e_i| \ll q$.
- Encrypt($m \in R_p$): : Set g(x) as a random subset sum of $\{f_i(x)\}$. Output c(x)=m+g(x).
- Decrypt: c(s) = m+(p multiple). Reduce mod p.

- ADD and MULT: Add or multiply the ciphertext polynomials.

SWHE from RLWE [BV11a]

Bar-Ilan University
Dept. of Computer Science

- Parameters: p, q with gcd(q,p)=1, $R = Z[y]/(y^n+1)$, $R_p = Z_p[y]/(y^n+1)$, $R_q = Z_q[y]/(y^n+1)$.
- KeyGen: Secret = uniform $s \in R$. Public key: linear polys $\{f_i(x)\}$ s.t. $f_i(s) = pe_i$, $|e_i| \ll q$.
- Encrypt($m \in R_p$): : Set g(x) as a random subset sum of $\{f_i(x)\}$. Output c(x)=m+g(x).
- Decrypt: c(s) = m+(p multiple). Reduce mod p.

Set p = 1 mod 2n, so p has n primitive 2n-th roots of unity. Then, R_p splits.

Message m(y) in R_p has n "plaintext slots" for m's evaluations at primitive n-th roots of unity mod p.

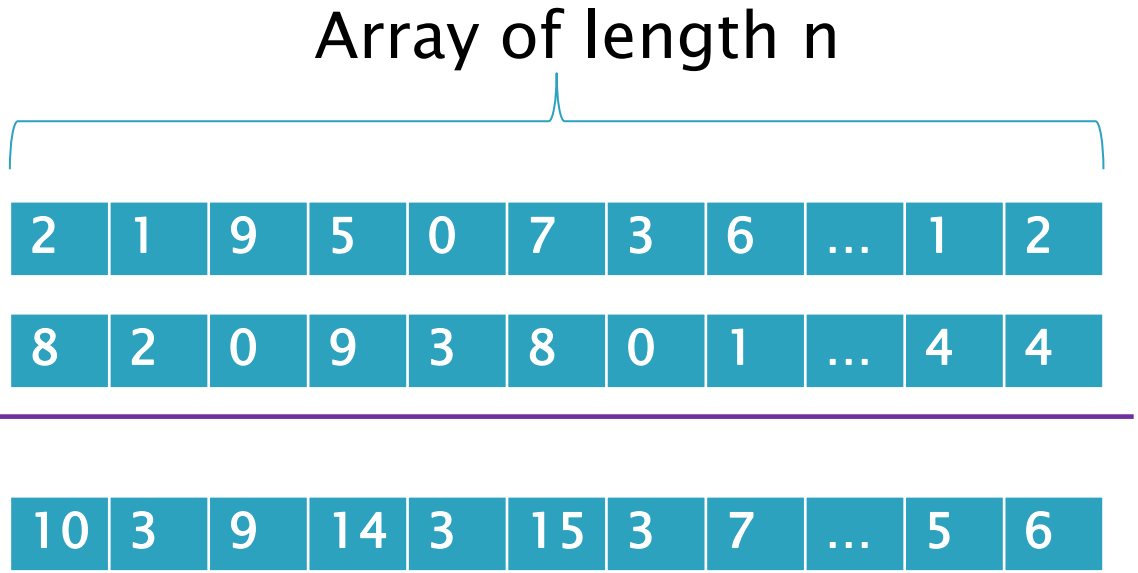
Forget Encryption for a Moment…

Bar-Ilan University
Dept. of Computer Science

- Plaintexts are $m(y) \in R_p = Z_p[y]/(y^n+1)$, represented by evaluations $m(\alpha_i)$, where α_i's are primitive n-th roots of unity mod p.
- $m_1(y)+m_2(y) \rightarrow m_1(\alpha_1)+m_2(\alpha_1),\ldots, m_1(\alpha_n)+m_2(\alpha_n)$.
- $m_1(y)\times m_2(y) \rightarrow m_1(\alpha_1)\times m_2(\alpha_1),\ldots, m_1(\alpha_n)\times m_2(\alpha_n)$.
- $F(m_1(y),\ldots,m_t(y))$
  $\rightarrow F(m_1(\alpha_1),\ldots,m_t(\alpha_1)), \ldots, F(m_1(\alpha_n),\ldots,m_t(\alpha_n))$.
- Compute F on n inputs $\{(a_{1i}, \ldots, a_{ti}) : i \in [n]\}$ *in parallel* by setting $\{m_i(y)\}$ so that $m_i(\alpha_j) = a_{ij}$.

SIMD (Single Instruction Multiple Data): Working on Data *Arrays*

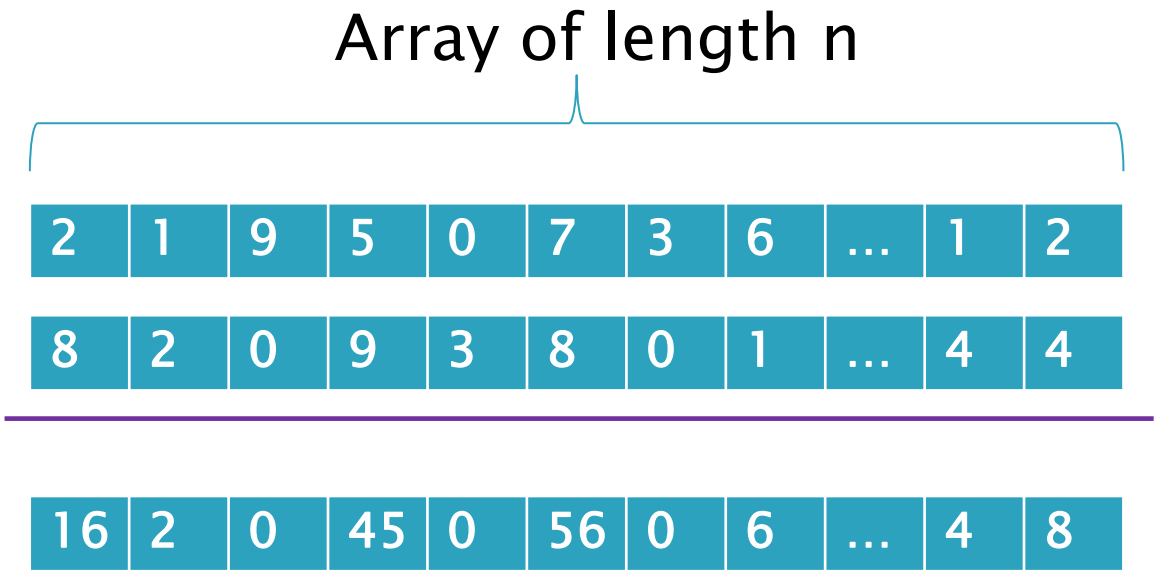
Bar-Ilan University
Dept. of Computer Science



n-ADD

SIMD (Single Instruction Multiple Data): Working on Data *Arrays*

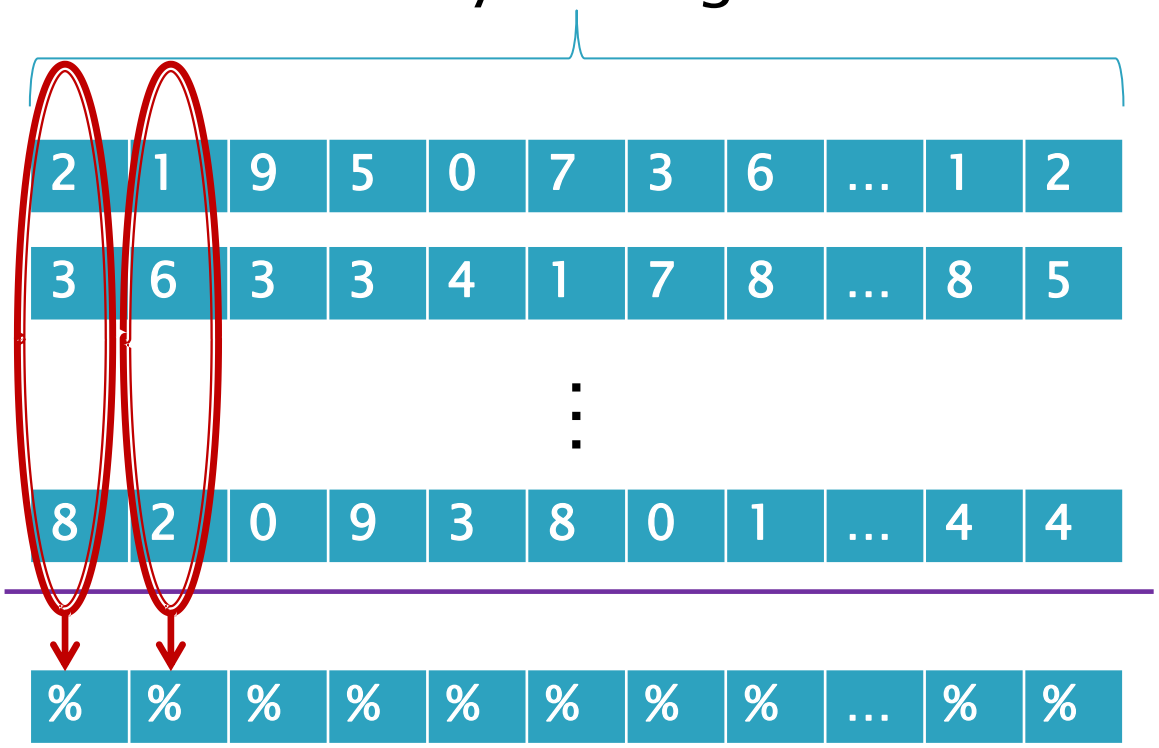
Bar-Ilan University
Dept. of Computer Science



n-MULT

SIMD (Single Instruction Multiple Data): Working on Data *Arrays*

Bar-Ilan University
Dept. of Computer Science



Function F

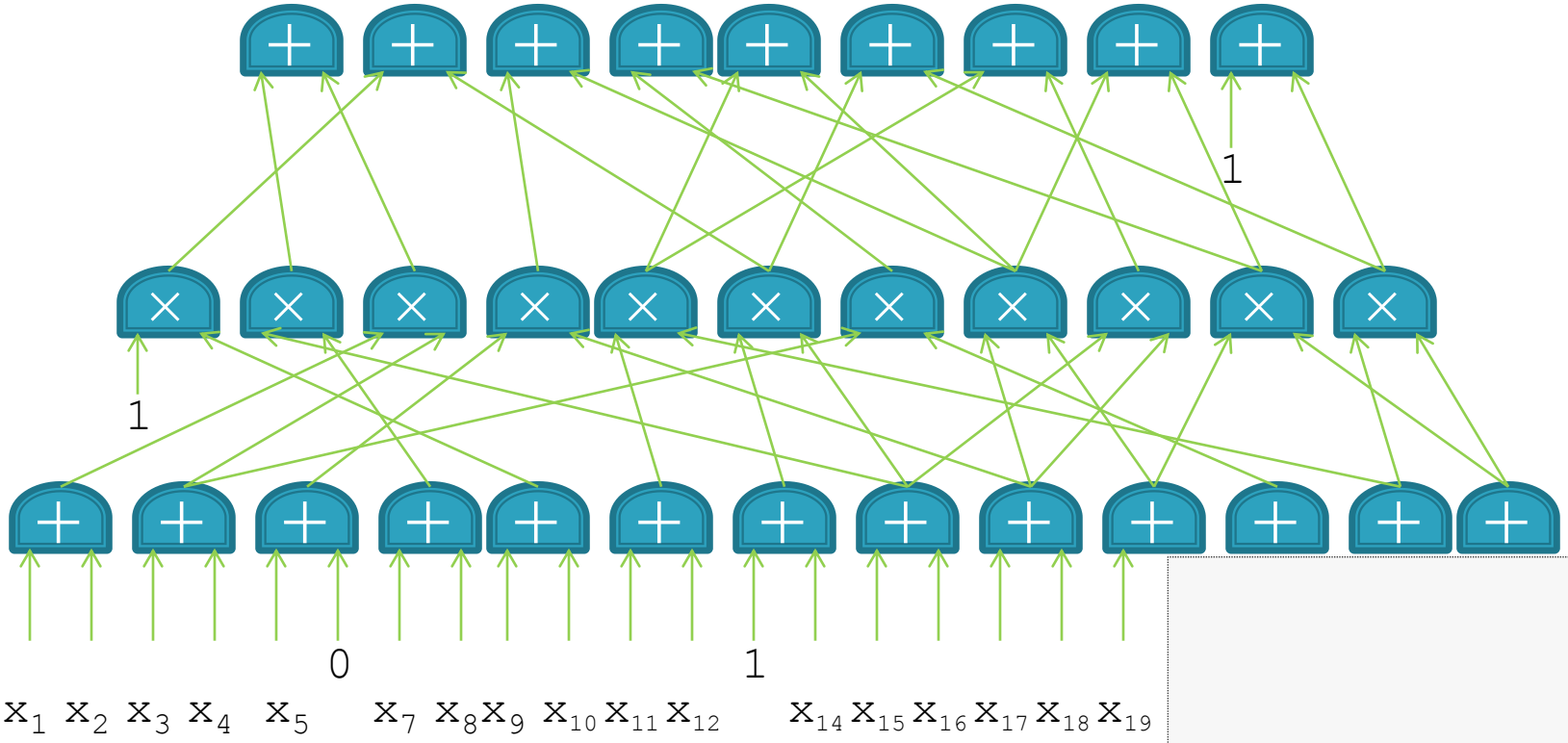
- Great for computing same function F on n different input strings.
- We can do SIMD homomorphically.

Bar-Ilan University
Dept. of Computer Science

More Complex Computation on Encrypted Arrays [GHS12]

So you want to compute some function… Not Using SIMD

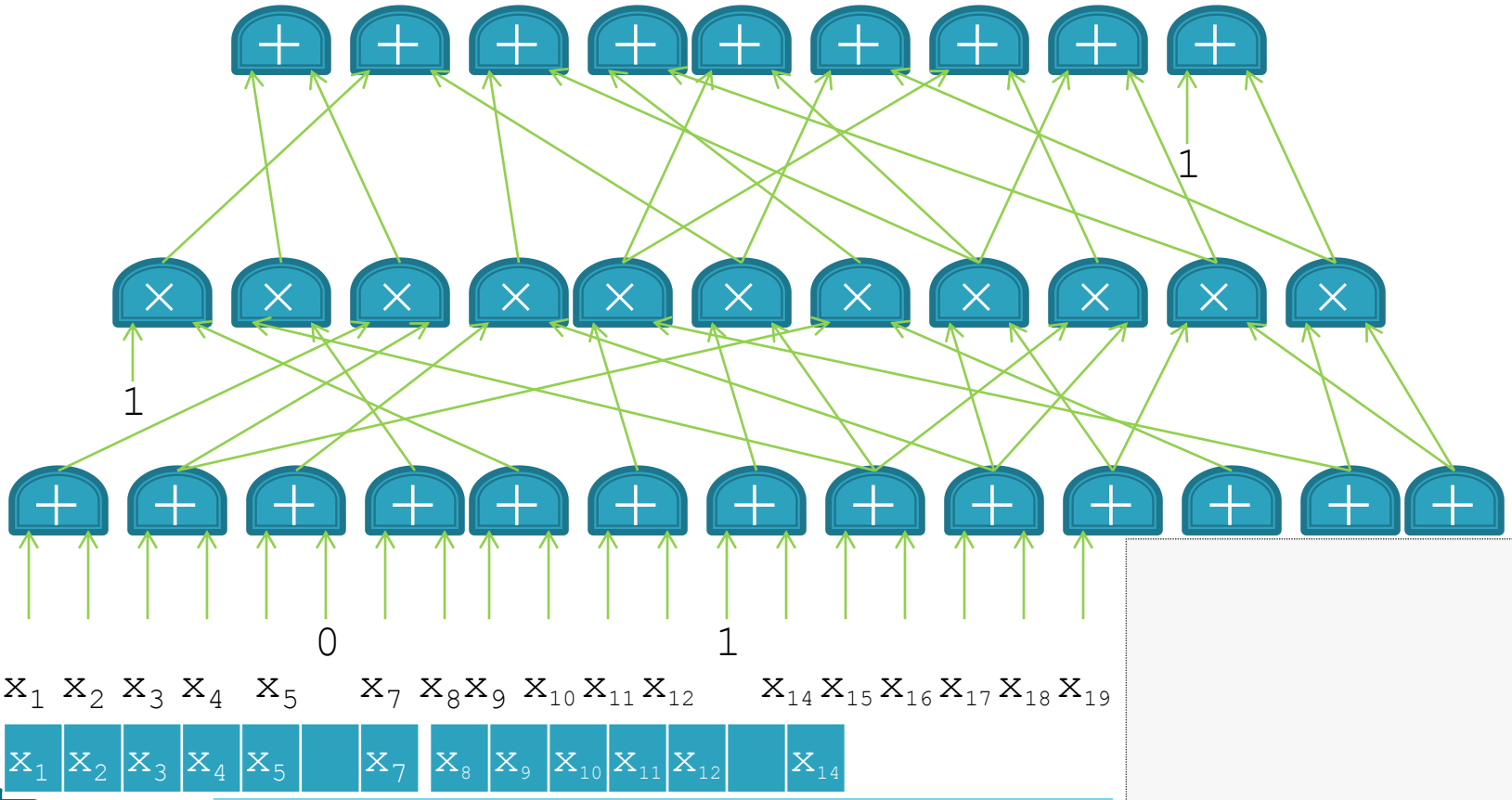
Bar-Ilan University
Dept. of Computer Science



ADD and MULT are a *complete* set of operations.

So you want to compute some function… Using SIMD…

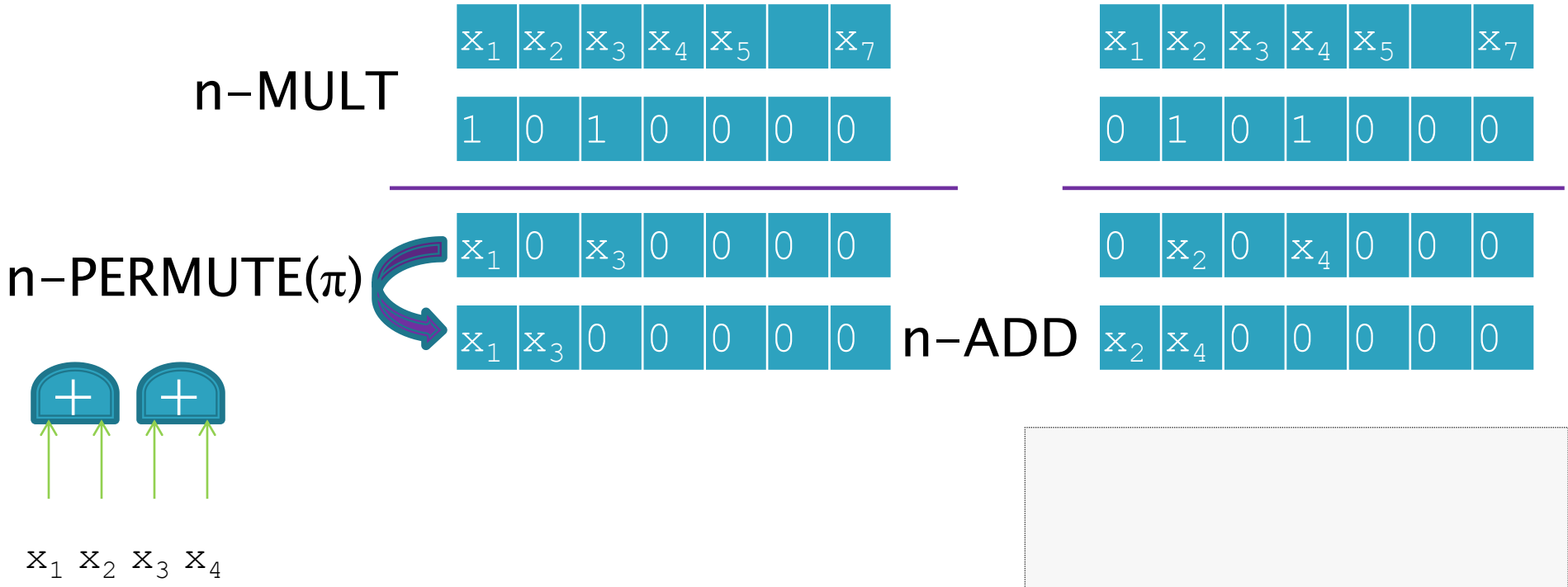
Bar-Ilan University
Dept. of Computer Science



n-ADD and n-MULT are *NOT* a *complete* set of operations.

n-ADD, n-MULT, n-PERMUTE: a complete set of SIMD ops

Bar-Ilan University
Dept. of Computer Science



Ring Automorphisms!

How do we Evaluate n-Permute(π) homomorphically, without "decompressing" the packed ciphertexts?

Ring Automorphisms

Bar-Ilan University
Dept. of Computer Science

Map $a(y) \rightarrow b(y) = a(y^i) \bmod (y^n+1)$, where $i \in Z_{2n}^*$.

$a(x) =$	$a(\alpha_1)$	$a(\alpha_2)$	$\cdots$	$a(\alpha_{n-1})$	$a(\alpha_n)$
$b(x) =$	$a(\alpha_1^i)$	$a(\alpha_2^i)$	$\cdots$	$a(\alpha_{n-1}^i)$	$a(\alpha_n^i)$
$=$	$a(\alpha_{\pi(1)})$	$a(\alpha_{\pi(2)})$	$\cdots$	$a(\alpha_{\pi(n-1)})$	$a(\alpha_{\pi(n)})$

b(y) has the same evaluations as a(y), but permuted!

Can We Evaluate These Automorphisms Homomorphically?

Bar-Ilan University
Dept. of Computer Science

- Given ciphertext $c_1(y)\cdot x + c_0(y)$ with
  $$c_1(y)\cdot s(y)+c_0(y) = m(y)+p\cdot e(y) \pmod{q, y^n+1}$$
- $c_1(y^i)\cdot s(y^i)+c_0(y^i) = m(y^i)+p\cdot e(y^i) \pmod{q, y^{in}+1}$, $i \in Z_{2n}^*$.
- $c_1(y^i)\cdot s(y^i)+c_0(y^i) = m(y^i)+p\cdot e(y^i) \pmod{q, y^n+1}$, $i \in Z_{2n}^*$.
- $c_1(y^i)\cdot x + c_0(y^i)$ is an encryption of $m(y^i)$ under key $s(y^i)$.
- Key switch $s(y)\rightarrow s(y^i)$ to get encryption of $m(y^i)$ under "normal" key $s(y)$.

Which Permutations Do the Automorphisms Give Us?

Bar-Ilan University
Dept. of Computer Science

- The "Basic" Permutations ($b(y) = a(y^i)$):
  - Only n (out of n!) of the possible permutations.
  - Automorphism group $Gal(Q(\alpha)/Q) \equiv Z_{2n}^*$.
  - Think of the automorphisms as n-ROTATE(i), which rotates the n items i steps clockwise, like a dial.
- Claim: For any permutation π, we can build n-PERMUTE(π) "efficiently" from n-ADD, n-MULT, and n-ROTATE(i).

n-PERMUTE(π) from n-ADD, n-MULT, and n-ROTATE(i). (Sketch)

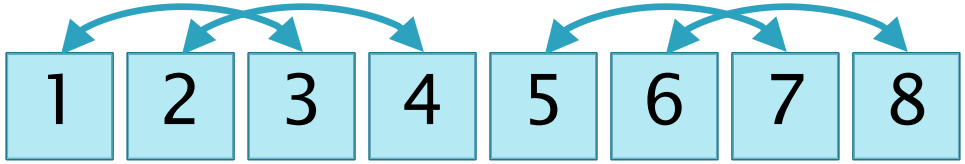
Bar-Ilan University
Dept. of Computer Science

- Butterfly network: assume $n = 2^k$.
  - n-PERMUTE(π) can be realized by a butterfly network of 2k-1 levels of n-SWAP(i,s) ops, $i \in \{1,\ldots,2k-1\}$, $s \in \{0,1\}^{n/2}$.
  - At level i, the 2^k items are partitioned into n/2 pairs, each pair with k-bit indices differing only in $|i-k|$-th bit.
  - n-SWAP(i,s) swaps the j-th pair iff $s_j=1$.

n-PERMUTE(π) from n-ADD, n-MULT, and n-ROTATE(i). (Sketch)

Bar-Ilan University
Dept. of Computer Science

- 8-SWAP(2,0110)

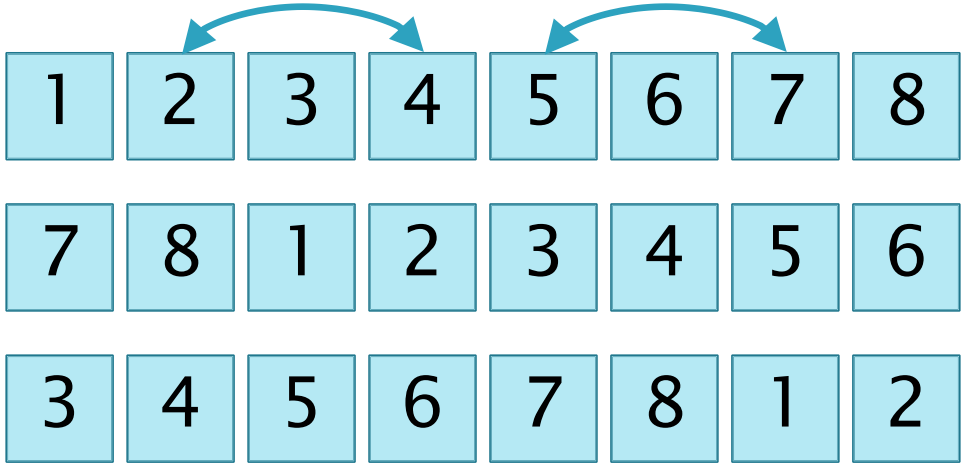


Potential Swaps

n-PERMUTE(π) from n-ADD, n-MULT, and n-ROTATE(i). (Sketch)

Bar-Ilan University
Dept. of Computer Science

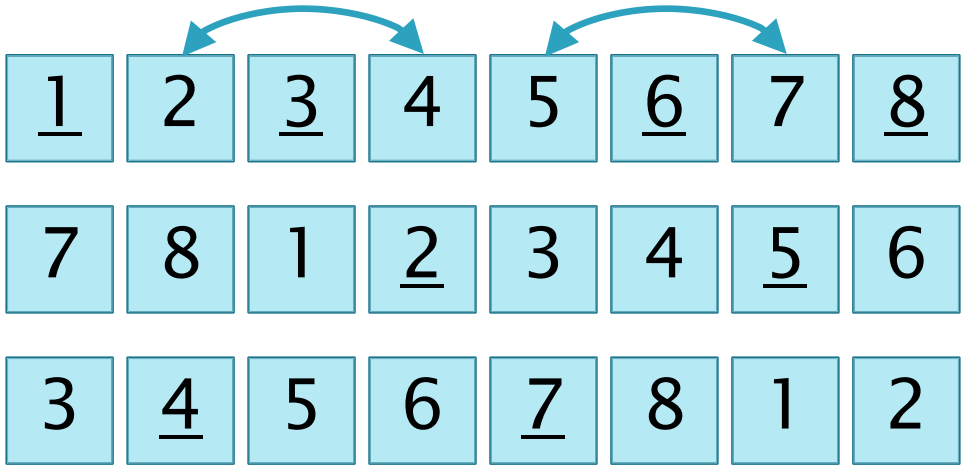
- 8-SWAP(2,0110)



n-PERMUTE(π) from n-ADD, n-MULT, and n-ROTATE(i). (Sketch)

Bar-Ilan University
Dept. of Computer Science

- 8-SWAP(2,0110)



Actual Swaps

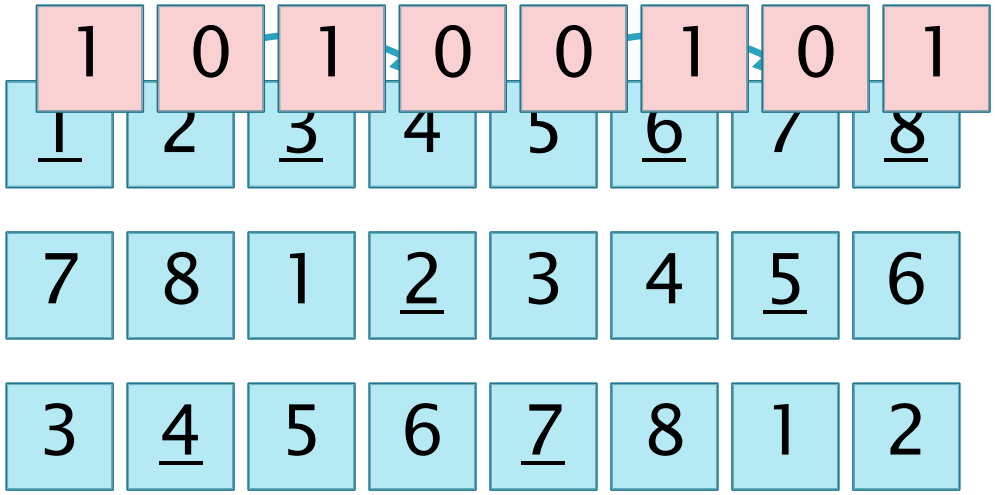
8-ROTATE(2)

8-ROTATE(-2)

n-PERMUTE(π) from n-ADD, n-MULT, and n-ROTATE(i). (Sketch)

Bar-Ilan University
Dept. of Computer Science

- 8-SWAP(2,0110)

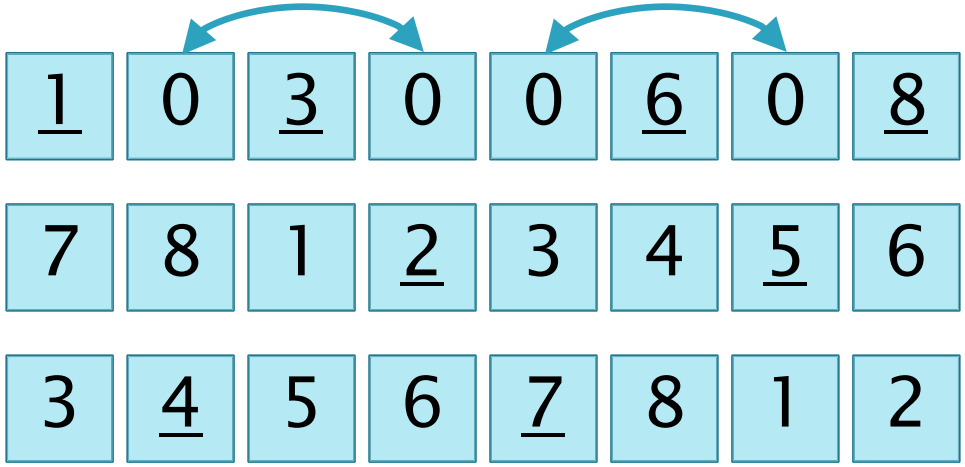


n-MULT

n-PERMUTE(π) from n-ADD, n-MULT, and n-ROTATE(i). (Sketch)

Bar-Ilan University
Dept. of Computer Science

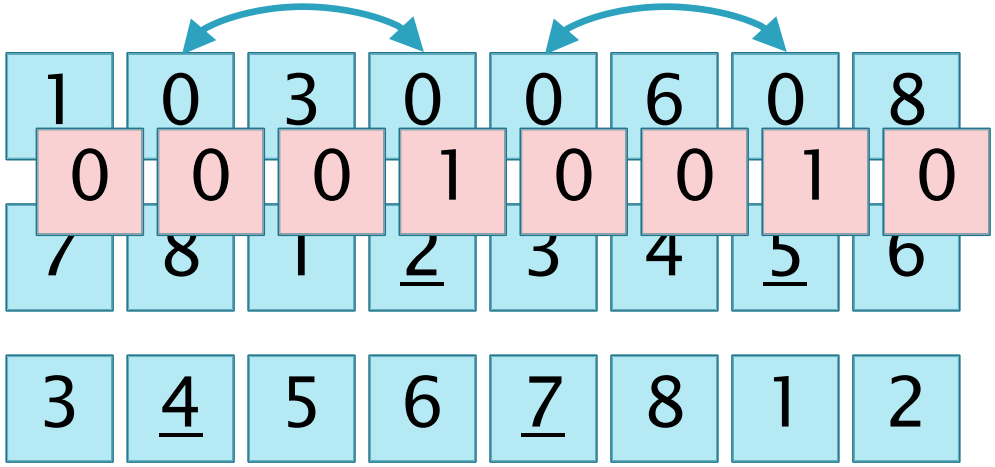
- 8-SWAP(2,0110)



n-PERMUTE(π) from n-ADD, n-MULT, and n-ROTATE(i). (Sketch)

Bar-Ilan University
Dept. of Computer Science

- 8-SWAP(2,0110)

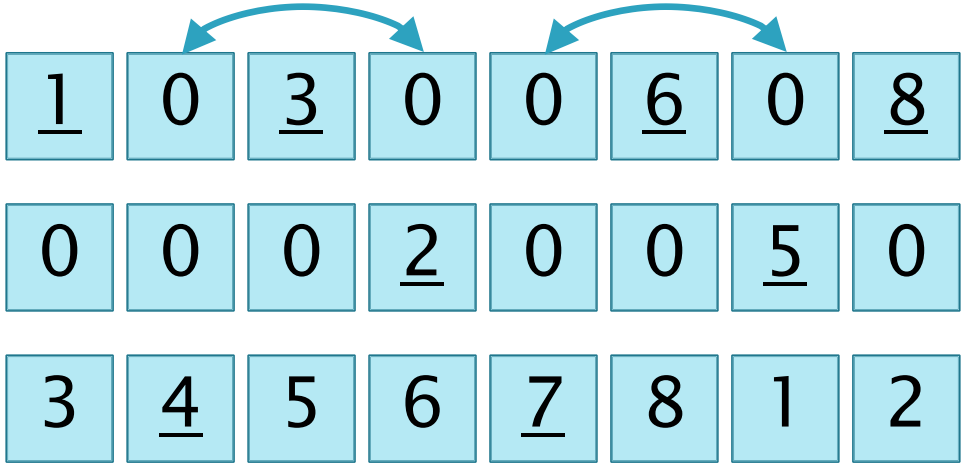


n-MULT

n-PERMUTE(π) from n-ADD, n-MULT, and n-ROTATE(i). (Sketch)

Bar-Ilan University
Dept. of Computer Science

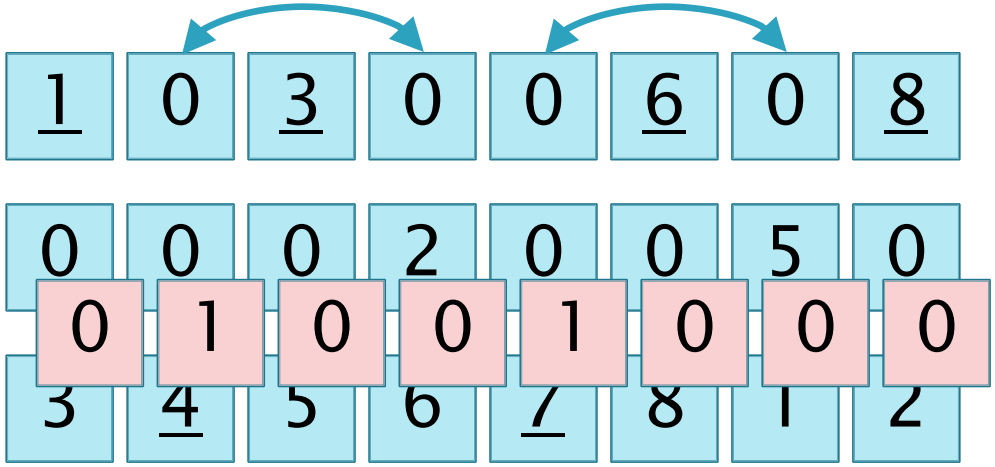
- 8-SWAP(2,0110)



n-PERMUTE(π) from n-ADD, n-MULT, and n-ROTATE(i). (Sketch)

Bar-Ilan University
Dept. of Computer Science

- 8-SWAP(2,0110)

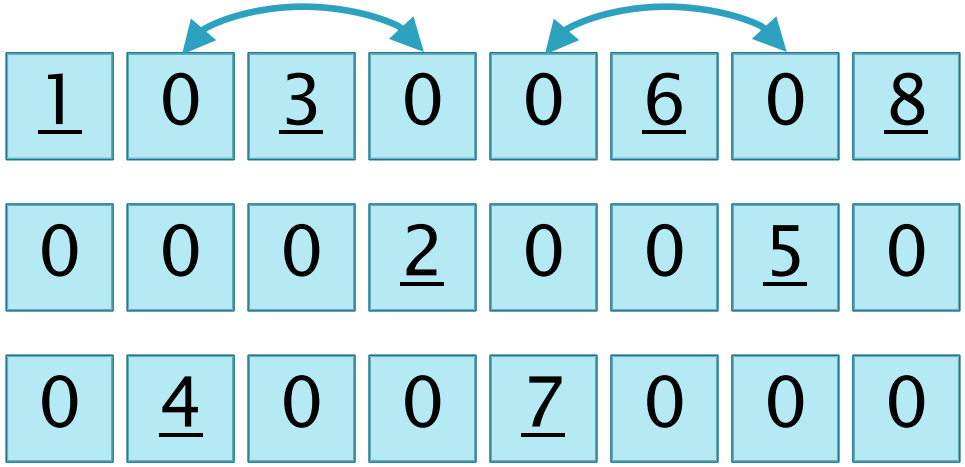


n-MULT

n-PERMUTE(π) from n-ADD, n-MULT, and n-ROTATE(i). (Sketch)

Bar-Ilan University
Dept. of Computer Science

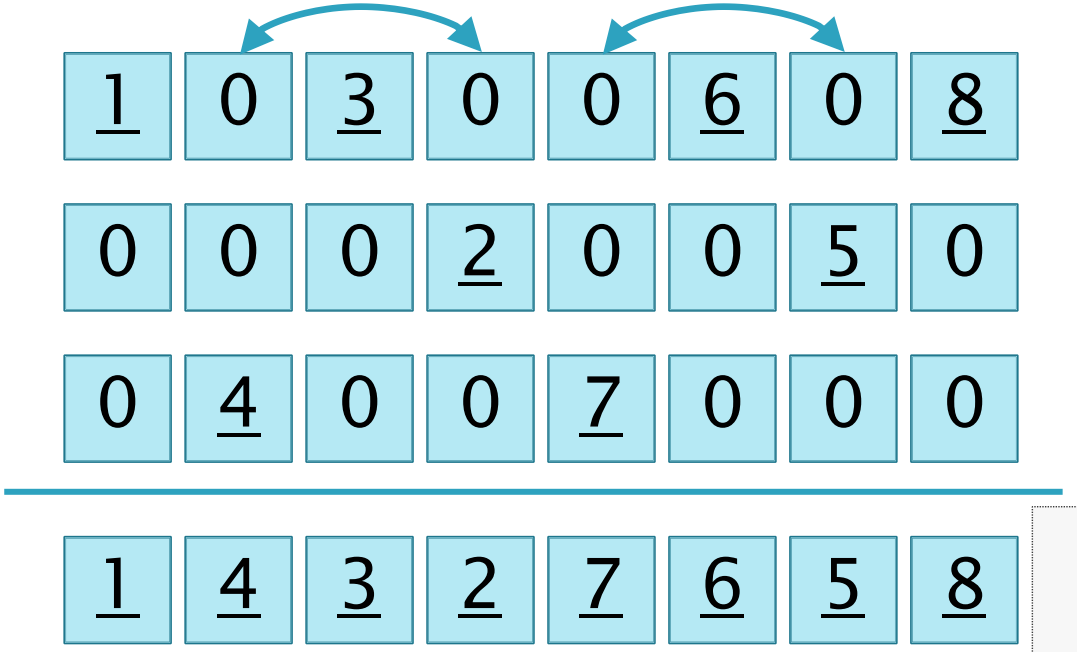
- 8-SWAP(2,0110)



n-PERMUTE(π) from n-ADD, n-MULT, and n-ROTATE(i). (Sketch)

Bar-Ilan University
Dept. of Computer Science

- 8-SWAP(2,0110)



n-ADD

Batching Summary

Bar-Ilan University
Dept. of Computer Science

- Overhead of batched RLWE-based BGV12 SWHE for security parameter k:
  = (encrypted comp. time)/(unencrypted comp. time)
  = poly(log q_L, log w) = poly(L, log k, log w), where w is the maximum width of circuit being evaluated.

Bar-Ilan University
Dept. of Computer Science

Fully Homomorphic Encryption [Gen09]

and the bootstrapping step…

Bootstrapping: What Is It?

Bar-Ilan University
Dept. of Computer Science

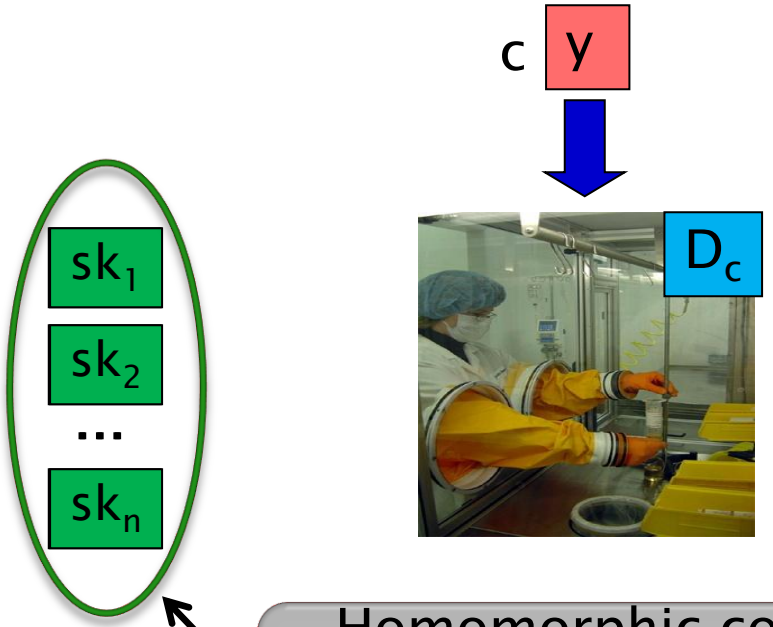
- So far, we can evaluate bounded depth F:

- 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?

Bar-Ilan University
Dept. of Computer Science

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



Bootstrapping in [BV11b,BGV12] (LWE-Based Scheme)

Bar-Ilan University
Dept. of Computer Science

- Recall: Complexity of BGV12 (and BV11b) decryption is independent of L, the depth it can evaluate.
- Set $L > 1+\text{depth needed to evaluate } D_C$.
- Then, homomorphic decryption reduces the noise level. (Use recursively.)
  - We now have FHE (modulo a circular security issue).

Bootstrapping BGV12

Bar-Ilan University
Dept. of Computer Science

- Decryption function computable in depth O(log k).
  - Our "somewhat homomorphic" scheme only needs to compute circuits of depth O(log k).
- BGV12 performance with bootstrapping:
  - Ciphertext size can be quasi-linear in k.
  - ADD and MULT take $\bar{O}(k)$ time.
  - Bootstrapping takes $\bar{O}(k^2)$ time.
    - Actually, with *batching*, we can reduce it to $\bar{O}(k)$ amortized.
  - Overhead is poly(L, log k, log w) = poly(log k, log w), where w is the maximum width of circuit being evaluated.
- Security can be based on quasi-polynomial factors: $2^{\bar{O}(\log^2 k)}$ versus 2^{k^c} (R)LWE.

Bar-Ilan University
Dept. of Computer Science

Chimeric FHE [GH11b]

A hybrid FHE scheme that combines lattices and Elgamal…

Main Idea

Bar-Ilan University
Dept. of Computer Science

Goal: Construct a bootstrappable SWHE scheme.

Problem

SWHE schemes don't handle multiplication well, it amplifies the "noisiness" of ciphertexts.

But Elgamal cannot alternate between additions and multiplications…

Solution?

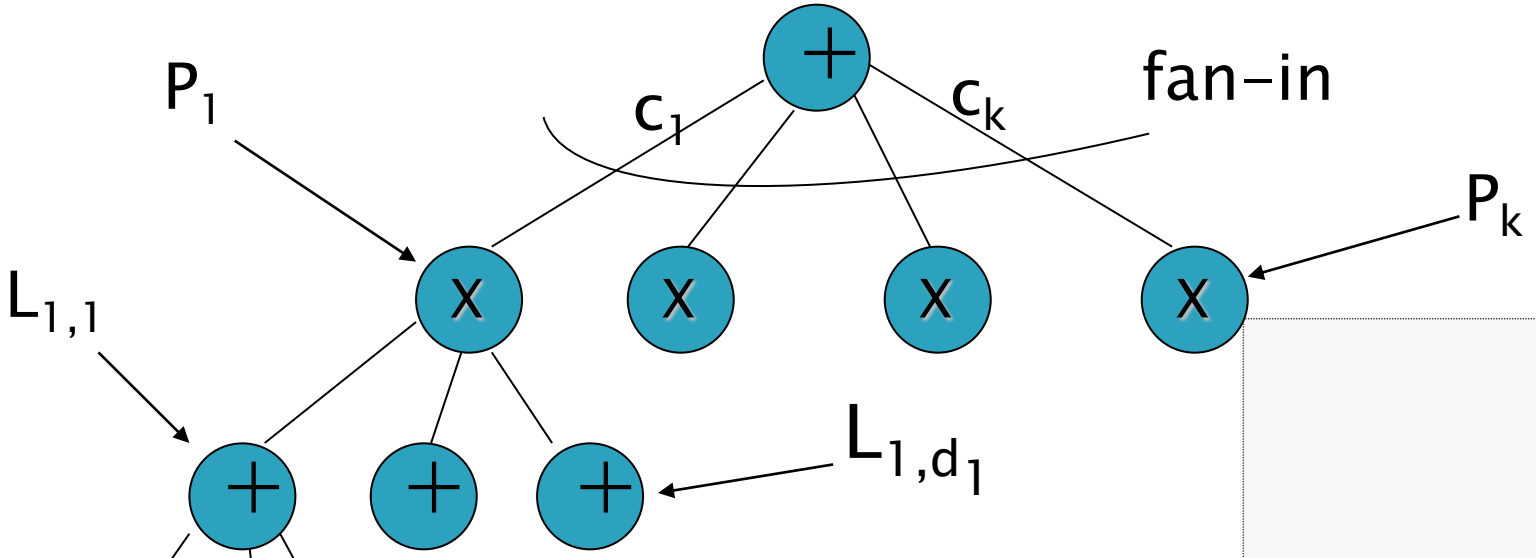
Elgamal handles multiplication well!
Maybe Elgamal can help!

Suppose the decryption function puts all of the mults *together*, without alternation? Can Elgamal help with the "product part"?

“Chimeric FHE”

Bar-Ilan University
Dept. of Computer Science

- SWHE: Use a lattice-based SWHE scheme, as before.
- Express SWHE decryption as a ciphertext-dependent depth-3 ($\Sigma\Pi\Sigma$) arithmetic circuit applied secret key.



$L_{i,j} = a_0 + \Sigma_{t=1\ldots n}\, a_t \cdot x_t$

$P_i = \Pi_{j=1\ldots d_i}\, L_{i,j}$

$C(\mathbf{x}) = \Sigma_{i=1\ldots k}\, c_i \cdot P_i = \Sigma_i\, c_i \cdot \Pi_j L_{i,j}$

"Chimeric FHE"

Bar-Ilan University
Dept. of Computer Science

- Bootstrapping: Evaluate depth-3 circuit homomorphically by combining a SWHE scheme with a "helper" MHE (multiplicative homomorphic enc.) scheme, like Elgamal:
  - Bottom Sums: Get MHE encryptions of the bottom sums.
    (Can put all needed MHE ciphertexts in public key.)
  - Products: Evaluate them homomorphically using MHE scheme.
  - Translation: Translate each ciphertext $Enc_{MHE}(m)$ to $Enc_{SWHE}(m)$ by evaluating MHE decryption homomorphically.
  - Top Sum: Evaluate top sum under the SWHE scheme.

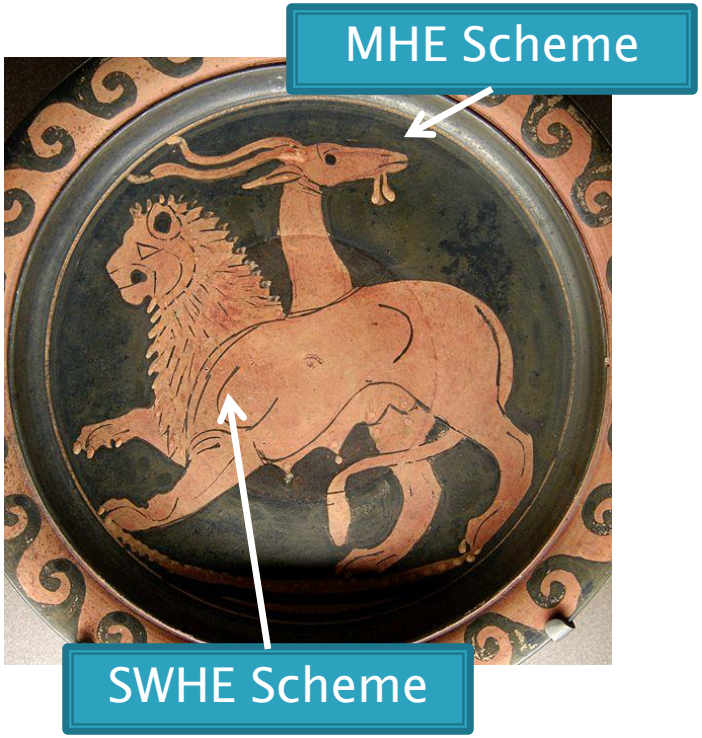
"Chimeric FHE"

Bar-Ilan University
Dept. of Computer Science

- Main high-level idea: The SWHE scheme only needs enough "homomorphic capacity" to evaluate the MHE scheme's decryption, not its *own* decryption.
  - Breaks the "self-referentiality" of bootstrapping.

“Chimeric” FHE?

Bar-Ilan University
Dept. of Computer Science



Chimera (mythology):

1) A monstrous fire-breathing female creature composed of the parts of multiple animals: upon the body of a lioness with a tail that ended in a snake’s head, the head of a goat arose on her back.
2) The term *chimera* has also come to mean, more generally, an impossible or foolish fantasy, hard to believe.

Bar-Ilan University
Dept. of Computer Science

Lattice-Based Decryption As a Depth-3 Circuit

Lattice-Based Decryption Functions

Bar-Ilan University
Dept. of Computer Science

- Typically, they can be computed using "restricted" depth-3 circuits.
- Proven already for Regev's cryptosystem by Klivans and Sherstov.

Elementary Symmetric Polynomials

Bar-Ilan University
Dept. of Computer Science

- Elementary symmetric polynomial $e_k(x_1, \ldots, x_n)$: sum of all monomials that are products of exactly k distinct variables.
- Cool fact: $e_k(\mathbf{x})$ mod p can be computed by a depth-3 arithmetic circuit (for large enough p)
- How? If $P(z) = \Pi_i(z+x_i)$, then $e_k(\mathbf{x})$ is the coefficient of z^{n-k}
- Computing P(z): evaluate P(z) in n+1
  - Let $A = \{a_1, \ldots, a_{n+1}\}$ be some subset of
  - Bottom Sums: Compute a_j+x_i for all x_i's and a_j's.
  - Products: Compute $\lambda_j \cdot P(a_j) = \Pi_i (a_j+x_i)$ for all j.
  - Top Sum: Interpolate $\sum_j \lambda_j \cdot P(a_j)$ to get desired coefficient of P(z).

Observe: the bottom sums are "restricted".

Multilinear Symmetric Polynomials

Bar-Ilan University
Dept. of Computer Science

- Multilinear symmetric polynomials (MSPs):
  - MSPs are symmetric, and each variable has degree 1
  - MSPs are linear combinations of elementary symmetric polynomials (ESPs)
  - MSPs can be computed by restricted depth-3 circuits.
- Lattice-based decryption functions can be expressed as “restricted” MSPs.

Bar-Ilan University
Dept. of Computer Science

An Elgamal Instantiation

Elgamal Example

Bar-Ilan University
Dept. of Computer Science

- MHE scheme: Elgamal over QR(p)
  - $p = 2q+1$ be a safe prime
- SWHE scheme: Plaintext space = Z_p. Decryption is a restricted depth-3 arithmetic circuit over Z_p.

Elgamal Example

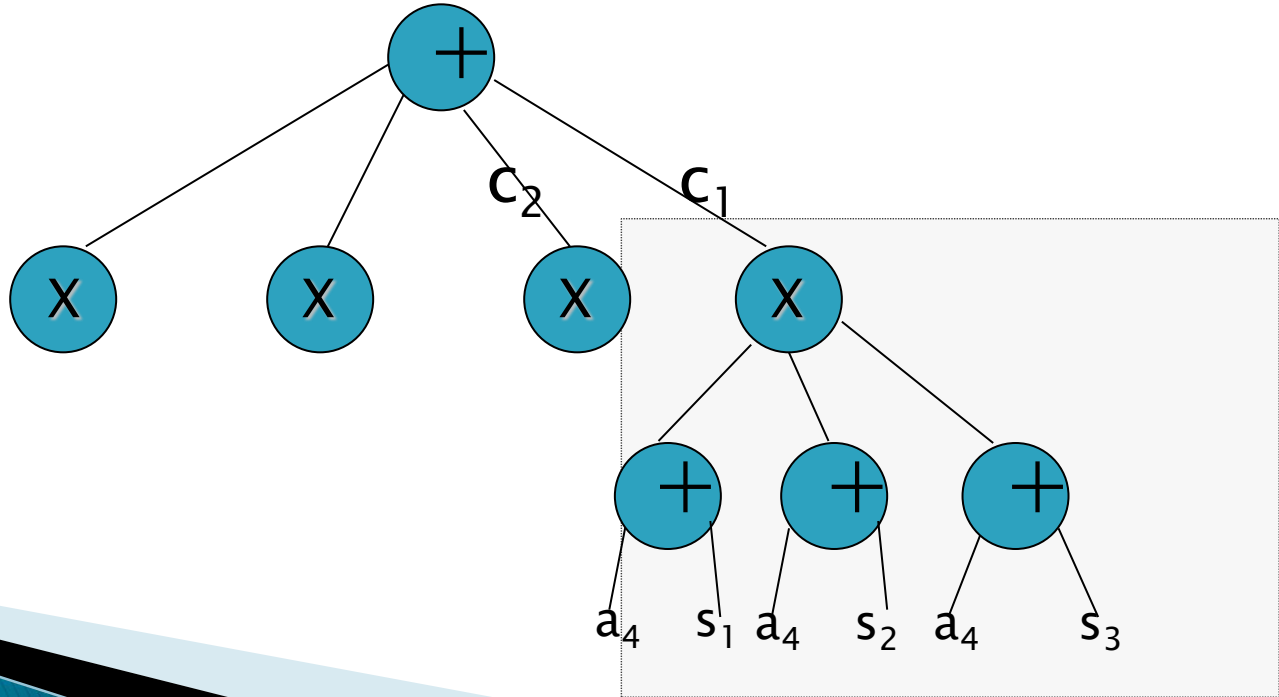
Bar-Ilan University
Dept. of Computer Science

- FHE.KeyGen:
  - Generate Elgamal key (e_L, g^{e_L}), SWHE key $(\{s_{iL}\}, pk_i)$, for every level L in the circuit
  - Encrypt individual bits of e_L under k_{L+1}.
  - Encrypt values a_j+s_i (in Z_p) under Elgamal for a_j in A.
    - Note: A is our set of "interpolation points" in our MSP.
    - Technicality: the a_j's must be chosen so that a_j and a_j+1 are both in QR(p), the plaintext space of Elgamal.
  - Publish the public keys and encrypted secret keys.

Elgamal Example: Refreshing a ciphertext

Bar-Ilan University
Dept. of Computer Science

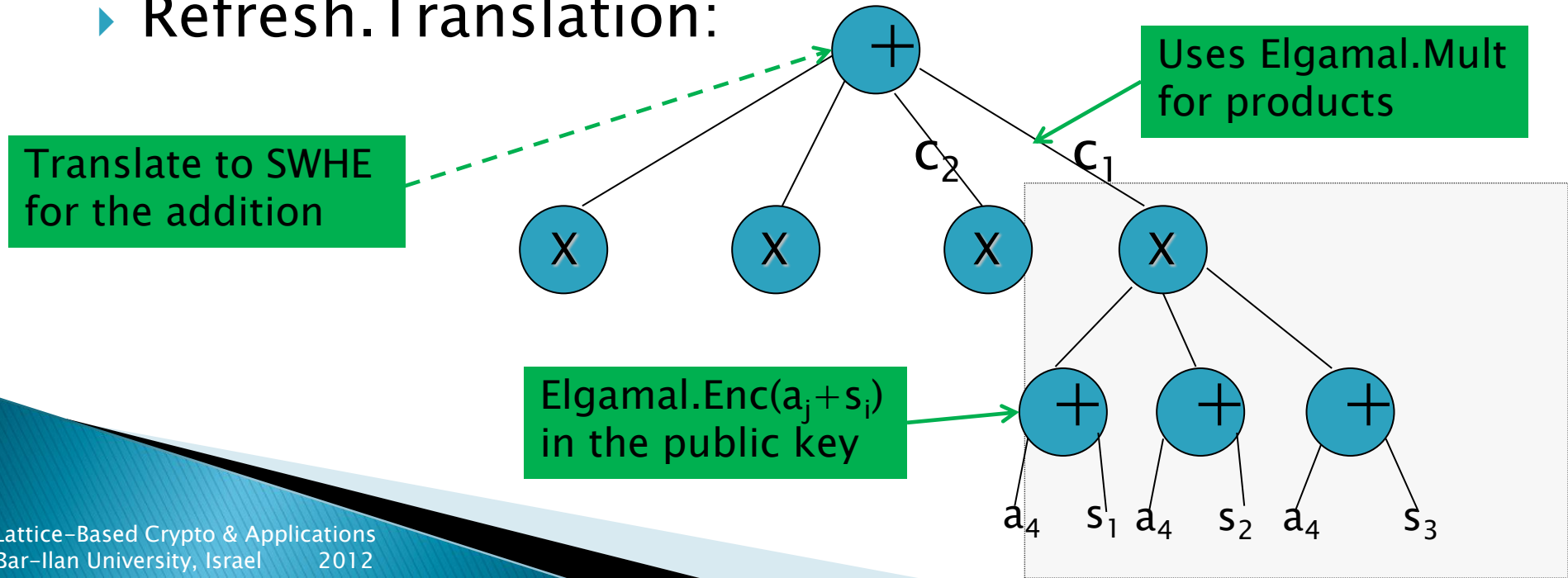
- To "refresh" a level-i ciphertext **c**:
  - First, express SWHE.Dec(**c**,**s**) as a **c**-dependent restricted depth-3 circuit taking key **s** as input.



Elgamal Example: Refreshing a ciphertext

Bar-Ilan University
Dept. of Computer Science

- Refresh.BottomSums:
  - Pick up the Elgamal encryptions of a_j+s_i from PK.
  - The bottom sums have been "precomputed".
- Refresh.Products:
  - Compute $c_j \cdot P(a_j) = c_j \cdot \Pi_i(a_j+s_i) \bmod p$ homomorphically using Elgamal.
- Refresh.Translation:



Elgamal Example (continued)

Bar-Ilan University
Dept. of Computer Science

- Refresh.Translation:
  - Goal: Convert $(y, z) = (g^r, mg^{-er})$ to a SWHE ciphertext.
  - Precompute $y_i = y^{2^i} \bmod p$ for all i up to log q.
  - "Inside" SWHE, compute $y^{e[i]2^i} = e[i] \cdot y^{2^i} + (1-e[i])y^0 \bmod p$.
  - Inside SWHE, compute product of $y^{e[i]2^i}$'s to get y^e.
    - The degree of this product is log q.
  - Inside SWHE, compute product of y^e and z to get m.
- Refresh.TopSum:
  - Just do it inside the SWHE scheme.

Elgamal Example (continued)

Bar-Ilan University
Dept. of Computer Science

- Required homomorphic capacity of SWHE scheme:
  - Evaluate Elgamal decryption, plus an ADD or MULT.
  - Overall degree = 2 log q.
  - Set SWHE parameters large to evaluate polynomials of degree 2 log q.
  - Done!

Thank You! Questions?

Bar-Ilan University
Dept. of Computer Science

Bibliography

Bar-Ilan University
Dept. of Computer Science

- [ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. Crypto 2009.
- [BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. ITCS 2012.
- [BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. Crypto 2011.
- [BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. FOCS 2011.
- [CNT12] Jean-Sebastien Coron, David Naccache, and Mahdi Tibouchi. Public-Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers. Eurocrypt 2012.
- [GH11b] Craig Gentry and Shai Halevi. Fully Homomorphic Encryption without Squashing Using Depth-3 Arithmetic Circuits. FOCS 2011.
- [GHS12] Craig Gentry, Shai Halevi, and Nigel Smart. Fully Homomorphic Encryption with Polylog Overhead. Eurocrypt 2012.
- [SV11] Nigel P. Smart, Frederik Vercauteren. Fully Homomorphic SIMD Operations. eprint.iacr.org/2011/133.

An Optimization

Bar-Ilan University
Dept. of Computer Science

- We can "compress" the entire FHE ciphertext down to a single MHE (e.g., Elgamal) ciphertext
- Choose a_j's cleverly so that all products $P(a_j)$ can be computed just from $P(a_1)$
  - Recall: $P(z) = \Pi_i (z+s_i)$ where s_i is a secret key bit.
  - We only "store" $P(a_1)$ – e.g., a single Elgamal ciphertext!
- Note: $P(a_j)$ can be computed homomorphically from $P(a_1)$ within the MHE scheme.
- Set a_j such that we know (w_j, e_j) such that
  - $a_j = w_j \cdot a_1^{e_j} \mod p$, and
  - $a_j+1 = w_j \cdot (a_1+1)^{e_j} \mod p$
  - How? Choose e_j and set
    $a_j = a_1^{ej}/((a_1+1)^{ej} - a_1^{ej})$ and $w_j = a_j/a_1^{ej}$.
- Then, $P(a_j) = w_j^d \cdot P(a_1)^{e_j} \mod p$

Twin Quadratic Residues

Bar-Ilan University
Dept. of Computer Science

Lemma: Let p be a prime. Let

$$S = \{(u,v): u \neq 0, v \neq 0, u^2 - v^2 = 1 \bmod p\}$$

Then, $|S| = p-3$ or $p-5$, depending on whether $p = 3$ or $1 \bmod 4$.

Proof: For each pair (u,v) in S, let $a_{uv} = u+v$. Then $a_{uv}^{-1} = u-v$, and we have:

$$u = (a_{uv} + a_{uv}^{-1})/2 \text{ and } v = (a_{uv} - a_{uv}^{-1})/2$$

implying that a_{uv} determines u and v uniquely. So, for

$$T = \{a \neq 0 : a+a^{-1} \neq 0, a-a^{-1} \neq 0\},$$

we have $|S| = |T|$.

We have that a is in T unless $a = 0$, $a^2 = -1$, or $a^2 = \pm 1$.
If $p = 1 \bmod 4$, then -1 in QR(p), and there are 5 prohibited values.
If $p = 3 \bmod 4$, then -1 is not a residue, and there are 3 prohibited values.