Homomorphic Secret Sharing

Elette Boyle — IDC

Niv Gilboa — BGU

Yuval Ishai — Technion & UCLA

1970

1980

1990

2000

2010

Primitives

PKE

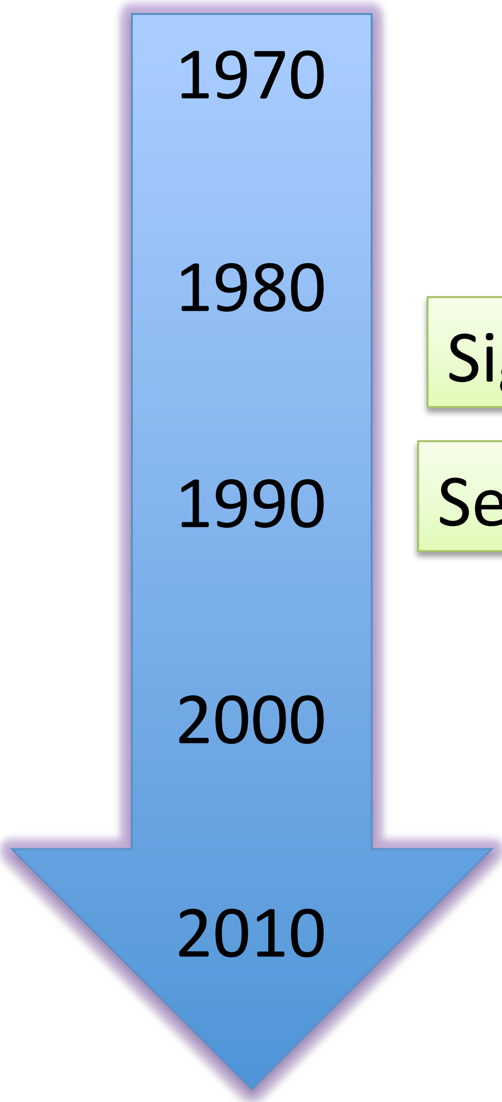
Signatures | ZK | OT

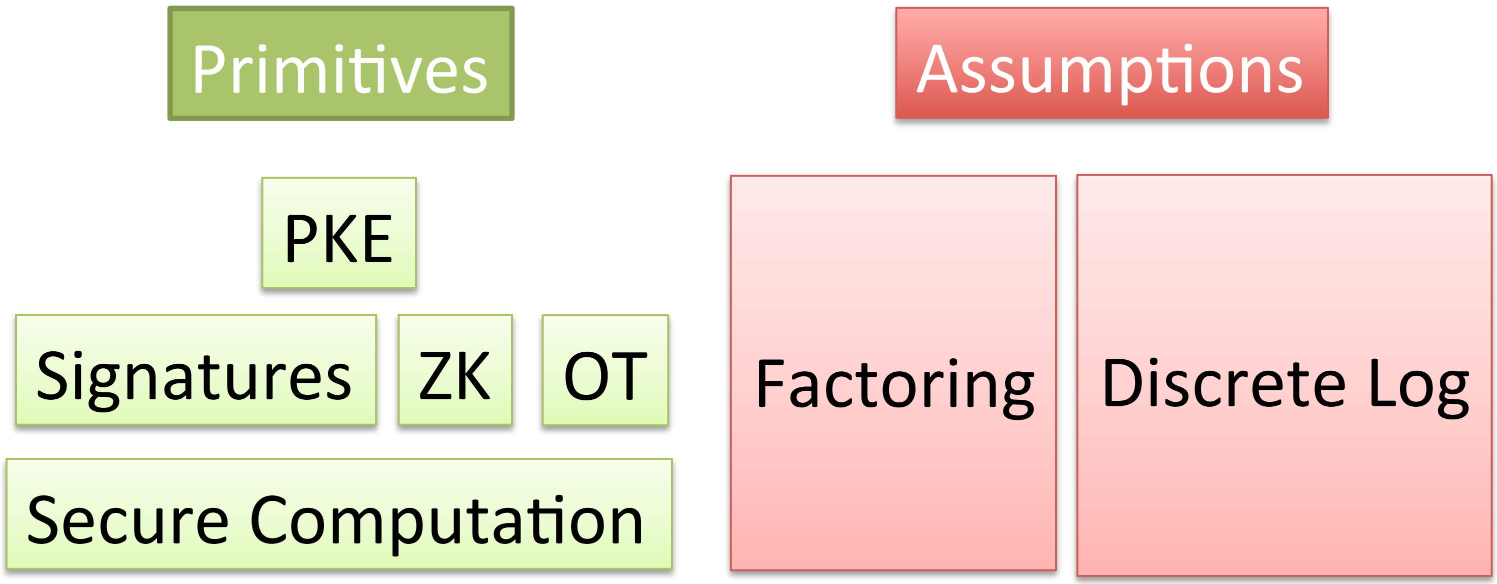
Secure Computation

Assumptions

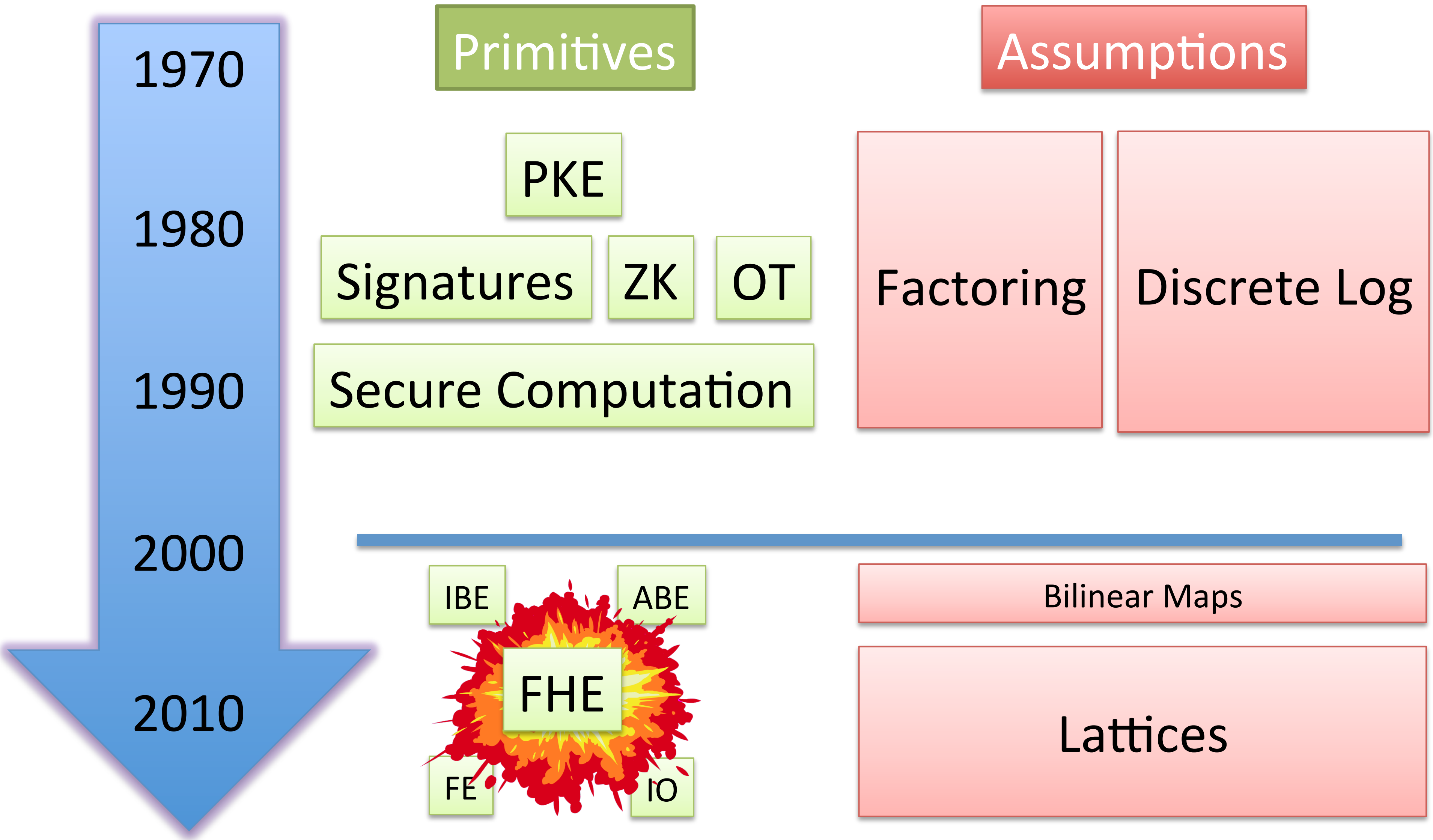
Factoring

Discrete Log





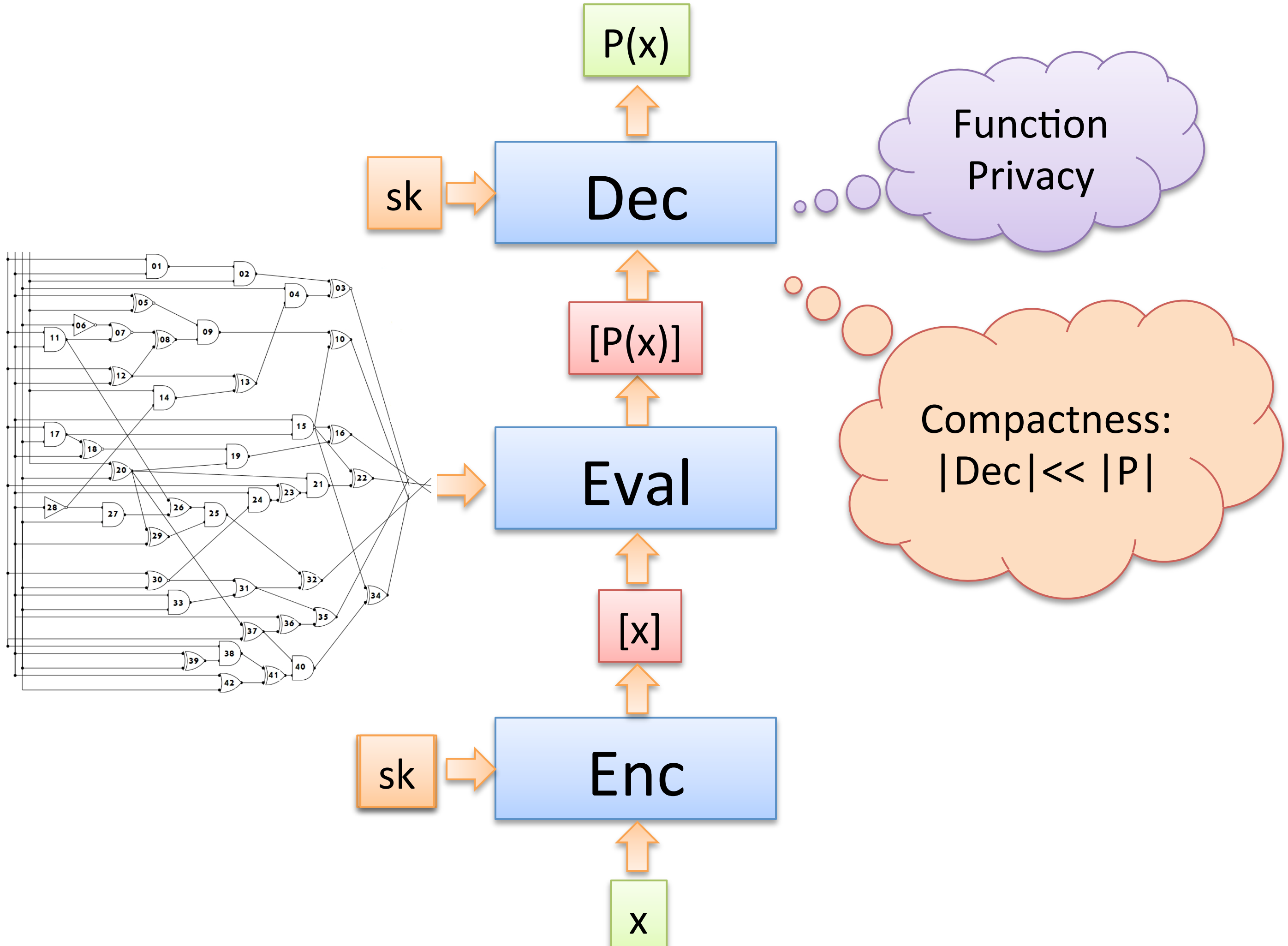
- Minimize communication?
- Minimize interaction?
- Minimize local computation?



1970
1980
1990
2000
2010
Primitives
PKE
Signatures
ZK
OT
Secure Computation
IBE
ABE
FHE
FE
IO
Assumptions
Factoring
Discrete Log
Bilinear Maps
Lattices

Fully Homomorphic Encryption

[RAD79,Gen09]

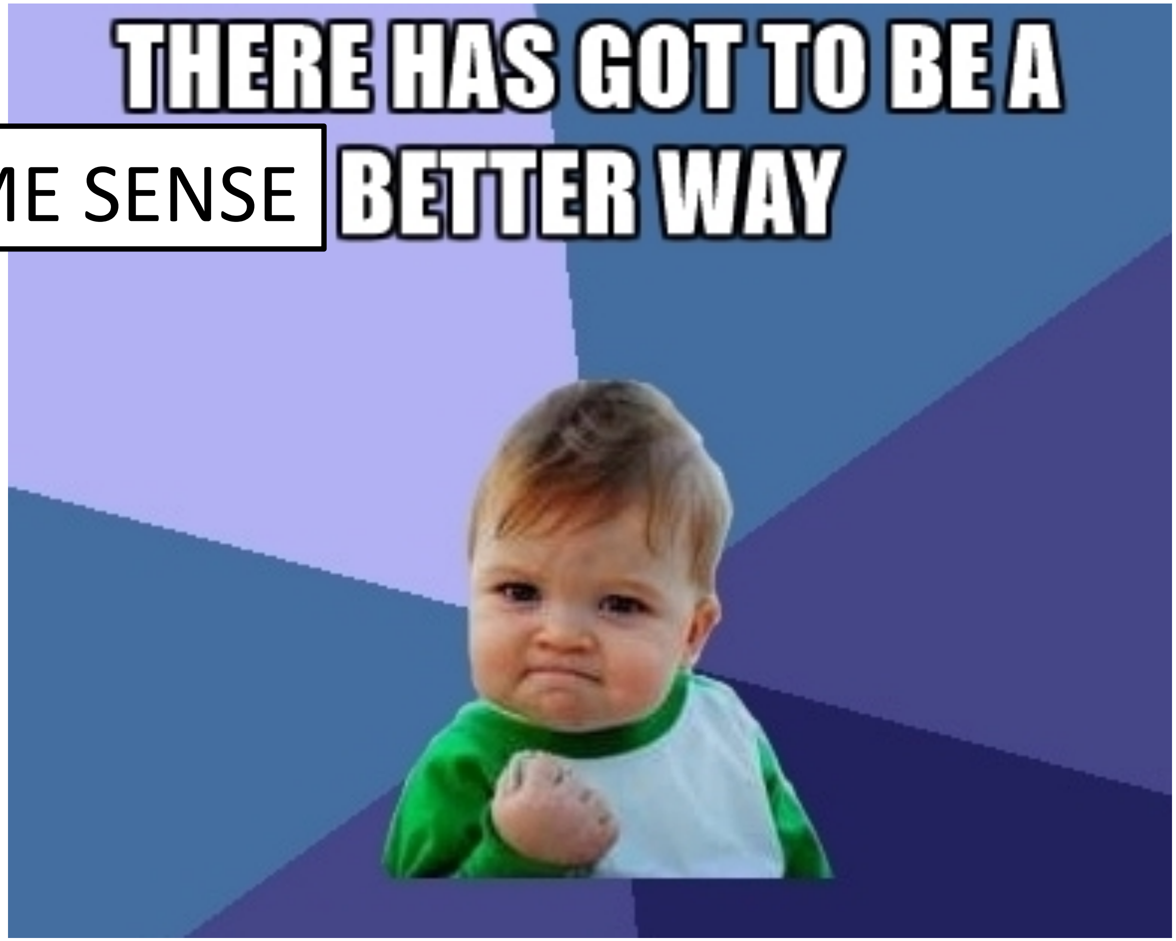


State of the FHE

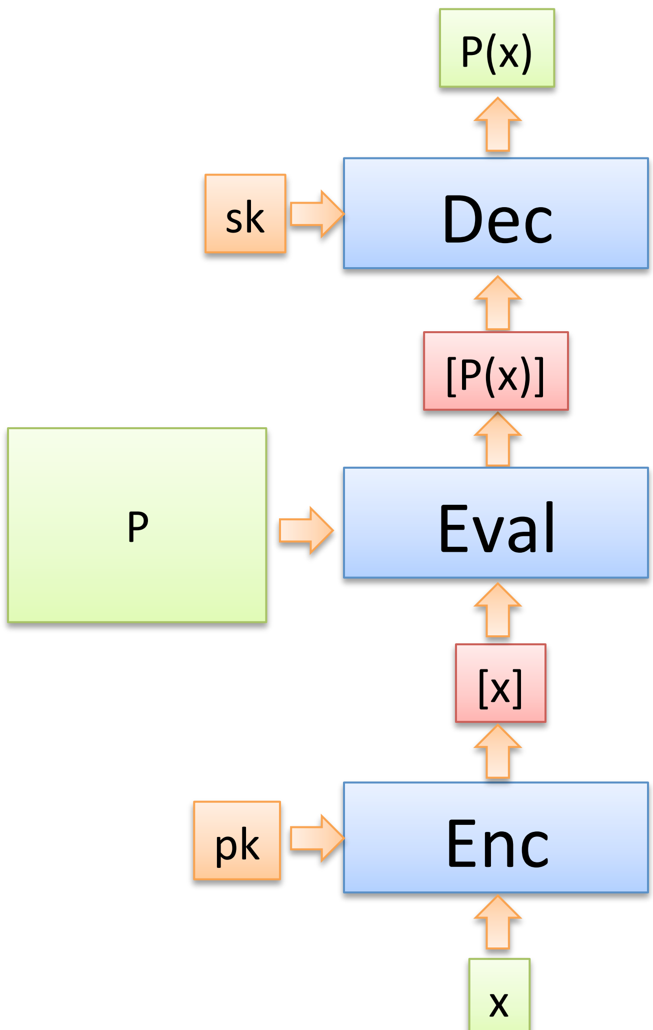
- The good
  - Huge impact on the field
  - Solid foundations [BV11,…]
  - Major progress on efficiency [BGV12,HS15,DM15,CGGI16]

> Given a generic group G:
> - Unconditionally secure PKE and even secure computation
> - Not known to be helpful for FHE

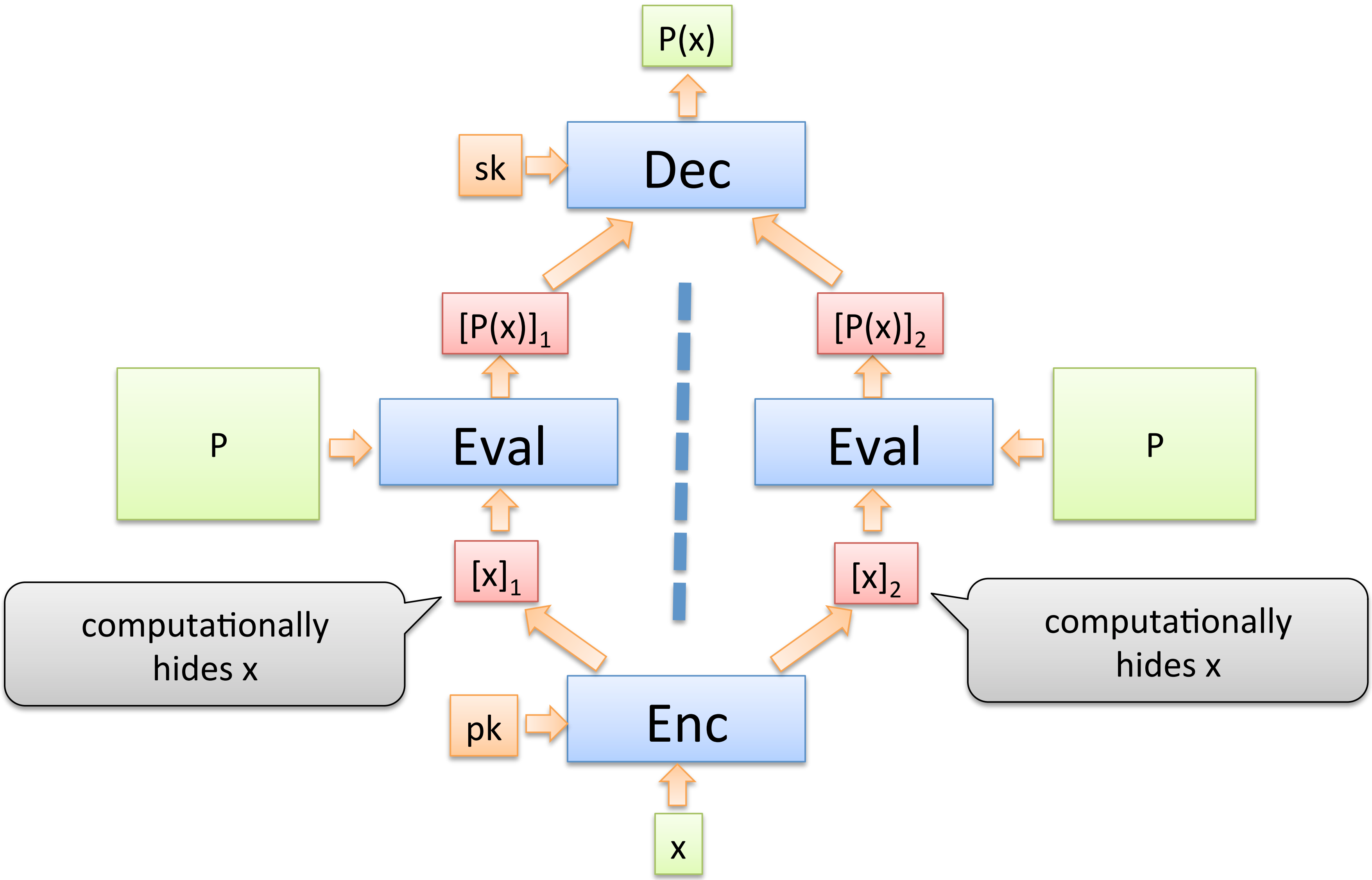
- The not so good
  - Narrow set of assumptions and underlying structures, all related to lattices
    - Susceptible to lattice reduction attacks and other attacks
  - Concrete efficiency still leaves much to be desired


THERE HAS GOT TO BE A
IN SOME SENSE
BETTER WAY

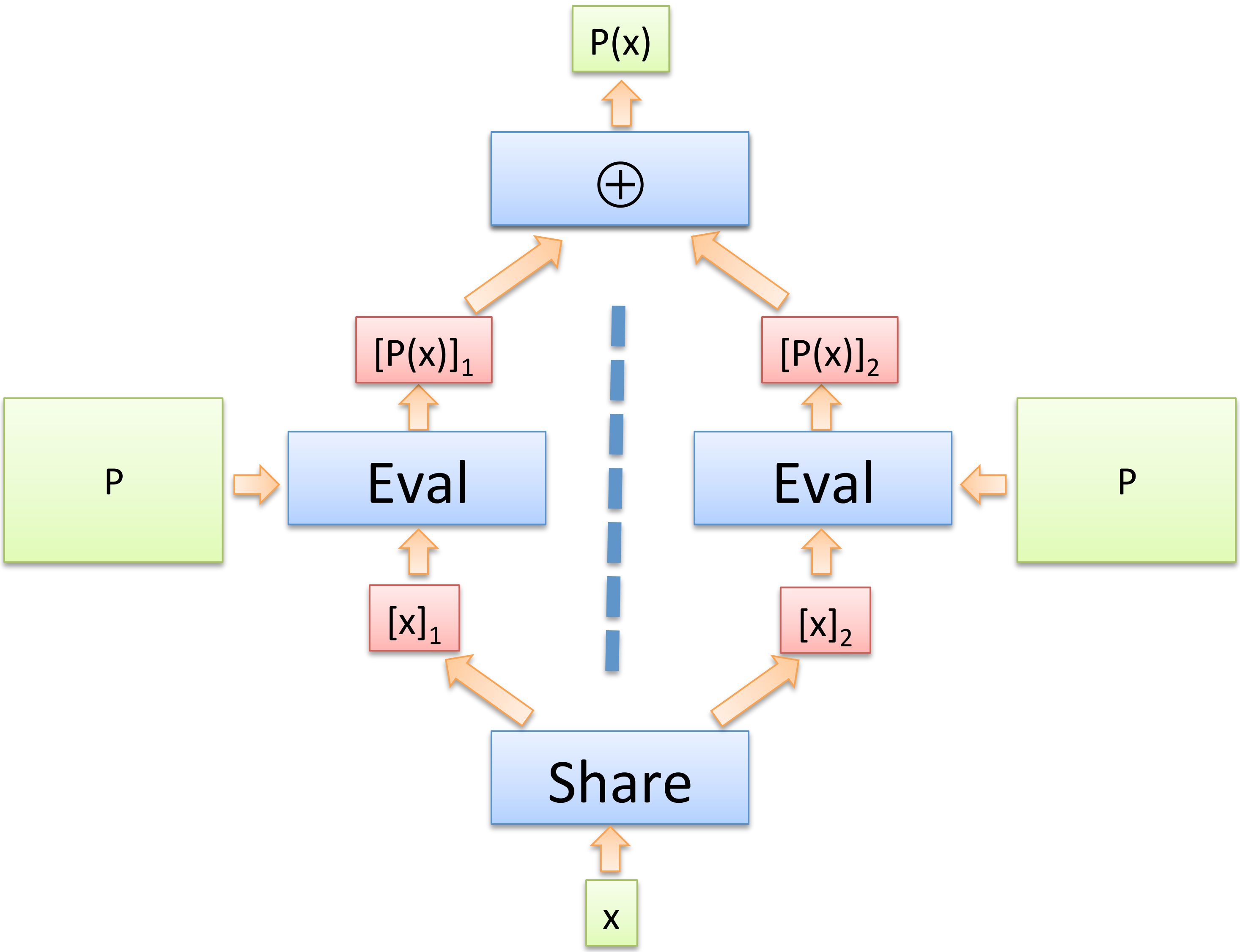
Recall: FHE



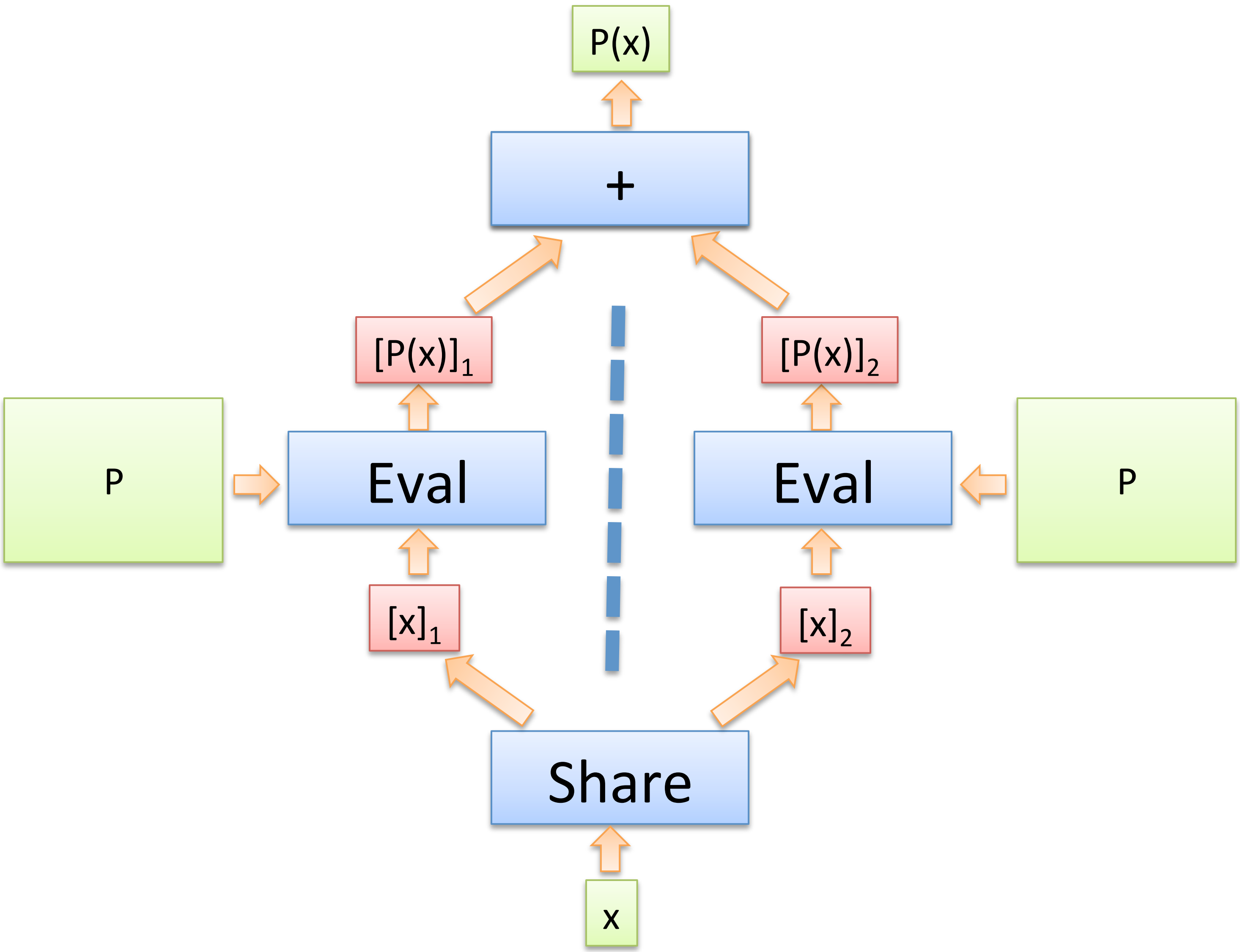
"1/2 FHE"


P(x)
sk
Dec
[P(x)]_1
[P(x)]_2
P
Eval
Eval
P
[x]_1
[x]_2
computationally hides x
computationally hides x
pk
Enc
x

(2-Party) Homomorphic Secret Sharing



(2-Party) Homomorphic Secret Sharing

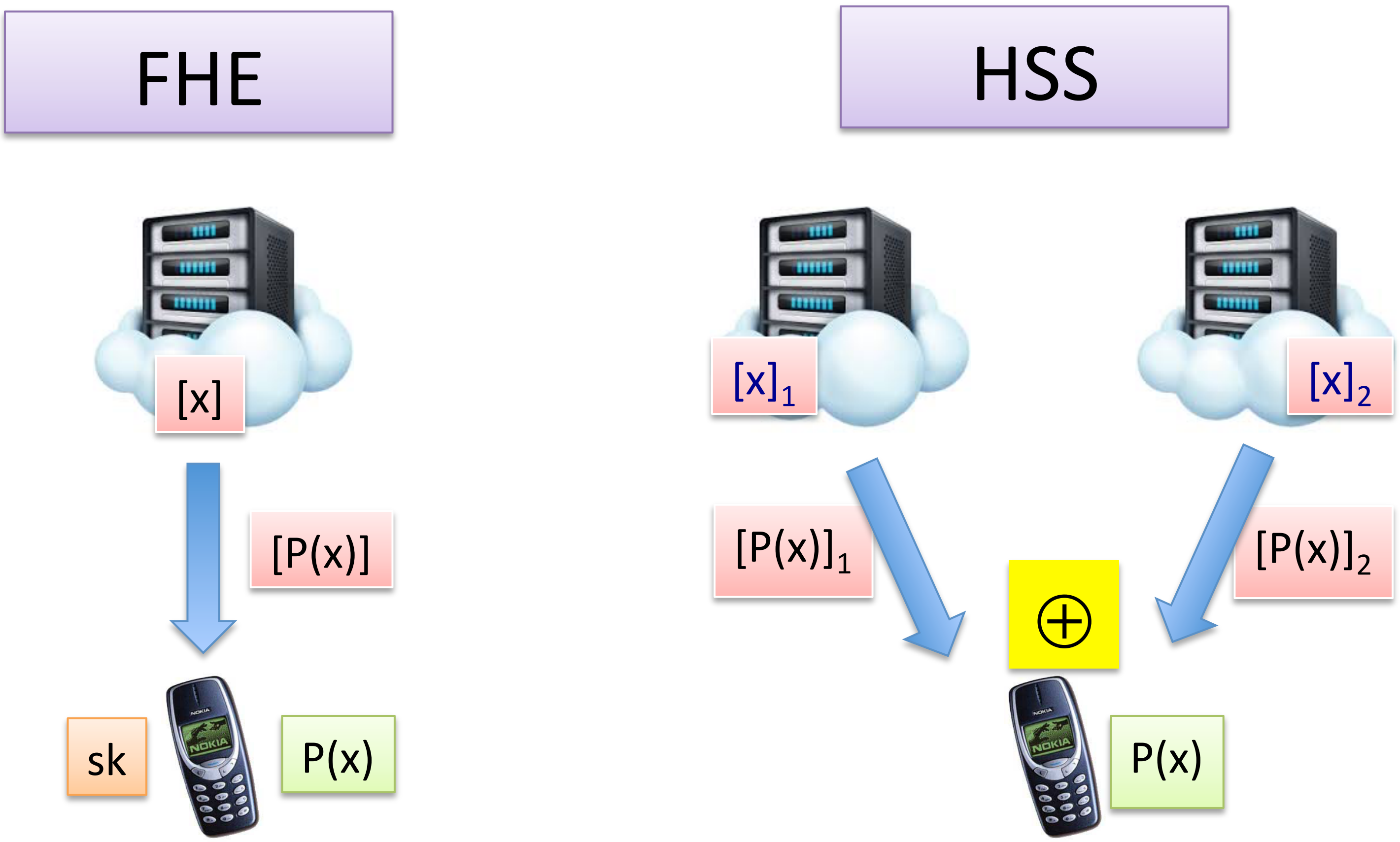


HSS vs. FHE

- HSS is generally weaker…
  - 2 (or more) shares vs. single ciphertext
  - Non-collusion assumption
- … but has some advantages
  - Ultimate output compactness
  - Efficient and public decoding
  - Can aggregate many outputs

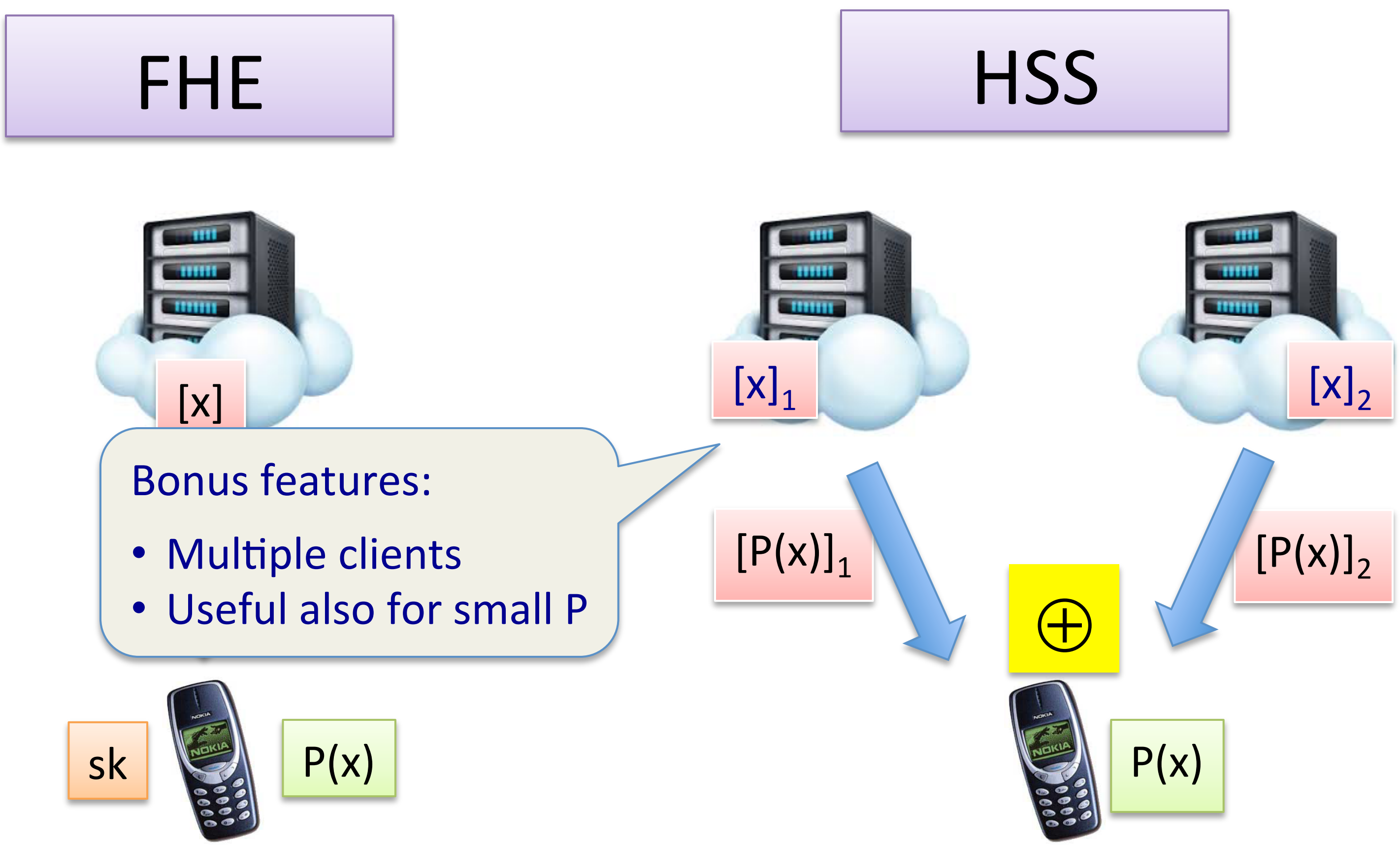
Applications

Delegating Computations to the Cloud



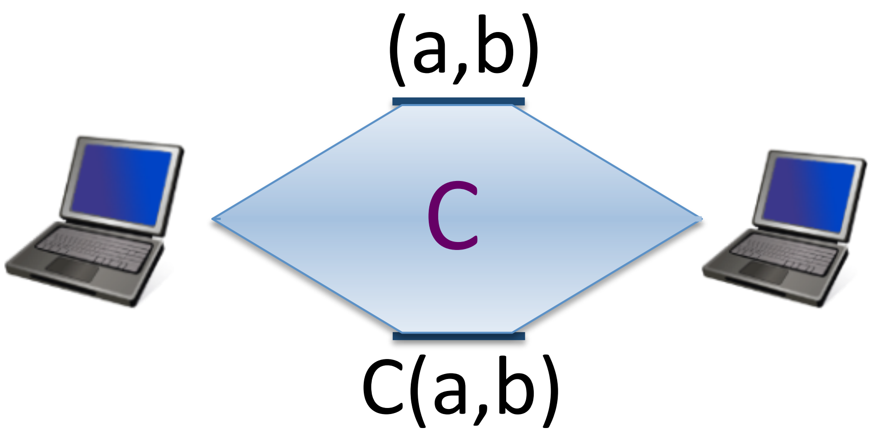
Applications

Delegating Computations to the Cloud



Applications

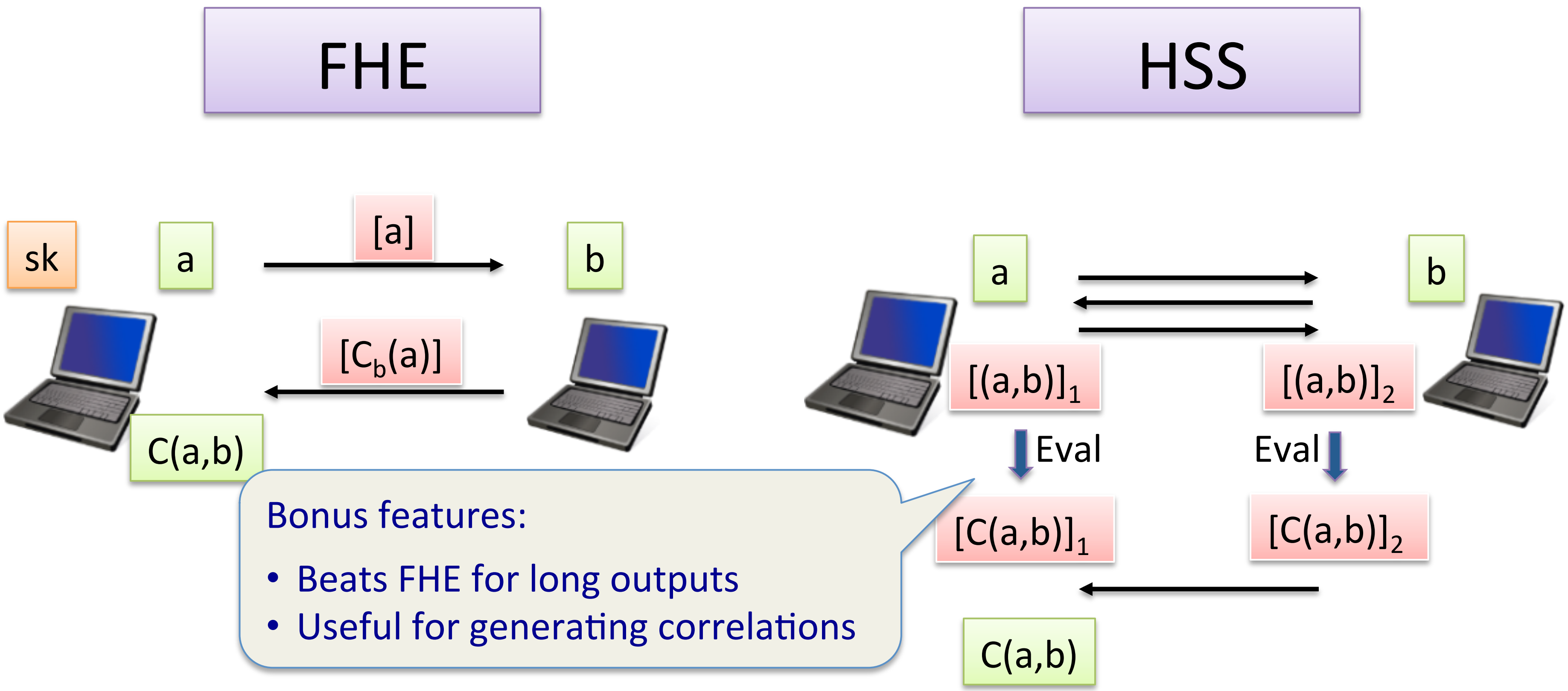
Communication complexity of securely computing C?



- Classically: > |C| [Yao86,GMW87,BGW88,CCD88,…]
  … even for restricted classes, such as formulas
- Using FHE: ~ |input|+|output|

Applications

Succinct Secure Computation



HSS for Circuits from LWE via FHE

- From multi-key FHE [LTV12,CM15,MW16,DHRW16]
  - "Additive-spooky" encryption [Dodis-Halevi-Rothblum-Wichs16]
- From threshold FHE [AJLTVW12,BGI15,DHRW16]

HSS without FHE?

20th century assumptions?

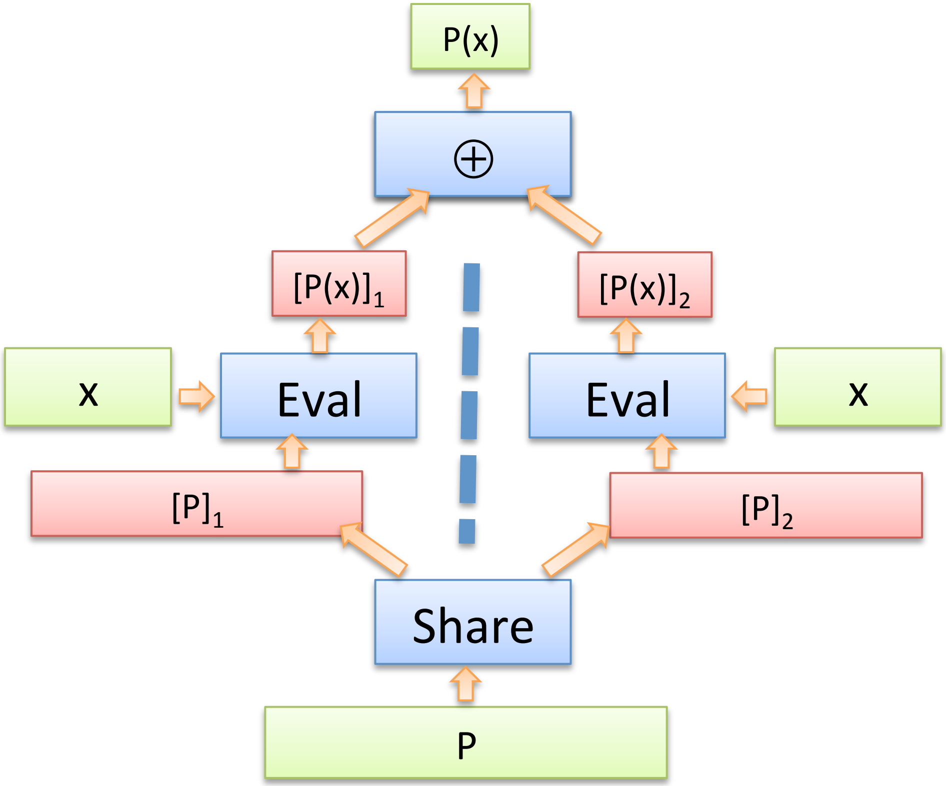
Coming Up

- HSS for “simple” functions from OWF
- HSS for branching programs from DDH
- Many open questions

Low-End HSS from OWF

Function Secret Sharing [BGI15]

- Reverse roles of function/program and input
- Share size can grow with program size



Function Secret Sharing [BGI15]

- Reverse roles of function/program and input
- Share size can grow with program size

- Very efficient constructions for "simple" classes from one-way functions [GI14,BGI15,BGI16]
  - Point functions
  - Intervals
  - Decision trees

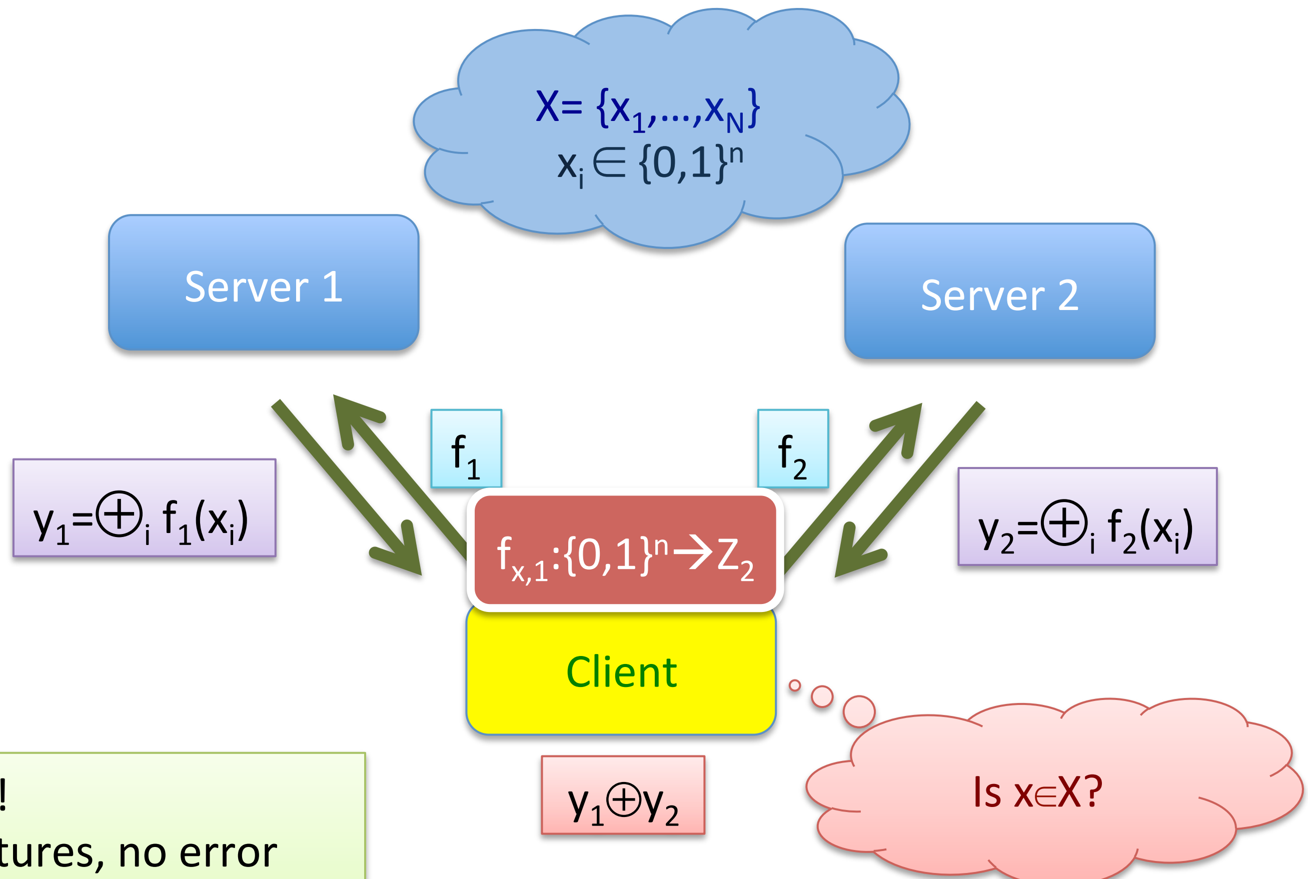
- Applications to privacy-preserving data access
  - Reading (e.g., PIR [CGKS95,CG97], "Splinter" [WYGVZ17])
  - Writing (e.g., private storage [OS98], "Riposte" [CBM15], "PULSAR" [DARPA-Brandeis])

Distributed Point Functions

- Point function $f_{\alpha,\beta}$:$\{0,1\}^n \rightarrow G$
  - $f_{\alpha,\beta}(\alpha)=\beta$
  - $f_{\alpha,\beta}(x)=0$ for $x \neq \alpha$
- DPF = FSS for class of point functions
  - Simple solution: share truth-table of $f_{\alpha,\beta}$
  - Goal: poly(n) share size
    - Implies OWF
  - Super-poly DPF implicit in PIR protocols [CGKS95,CG97]

Applications: Reading

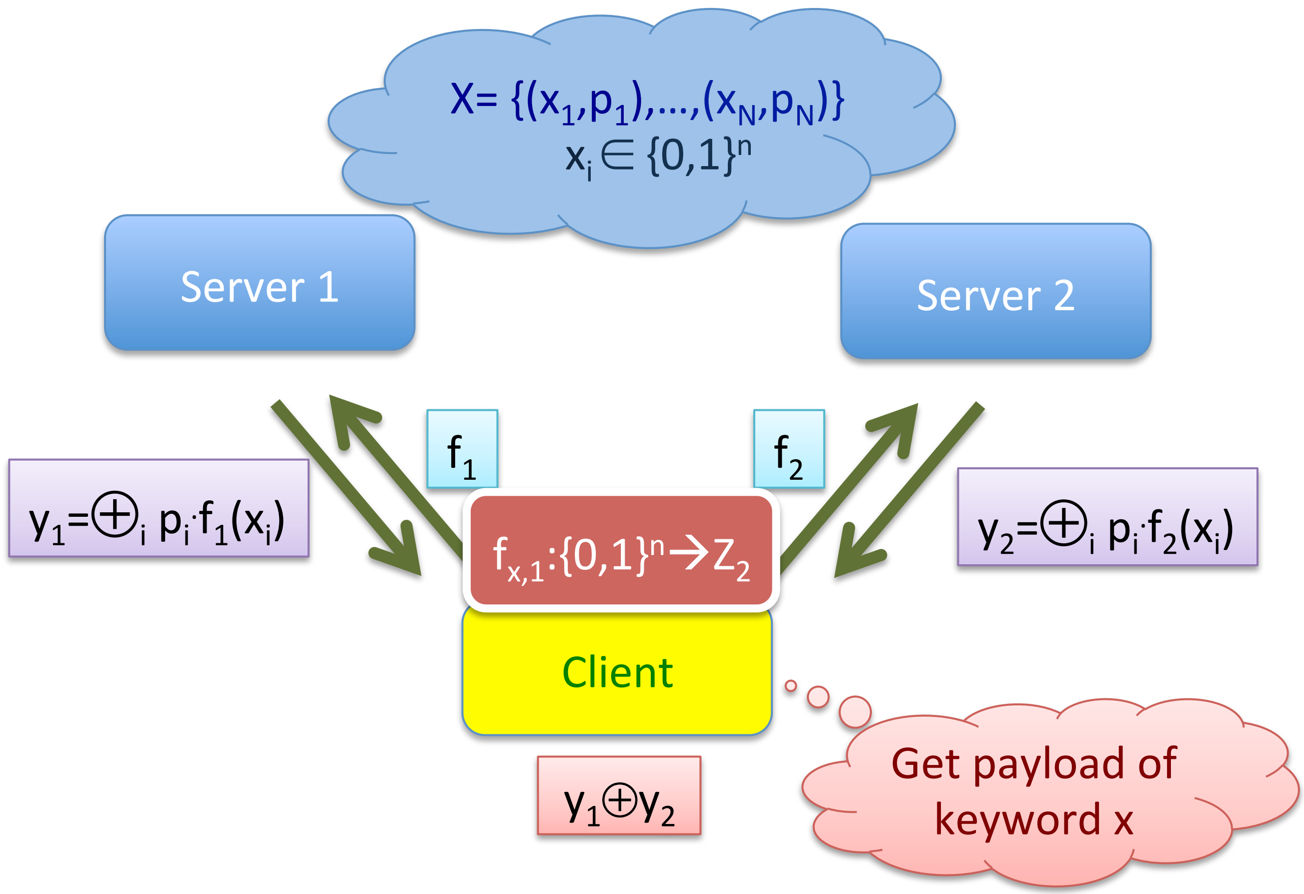
- Keyword search [CGN96,FIPR05,OS05,HL08, …]



1-bit answers!
No data structures, no error
Works well on streaming data

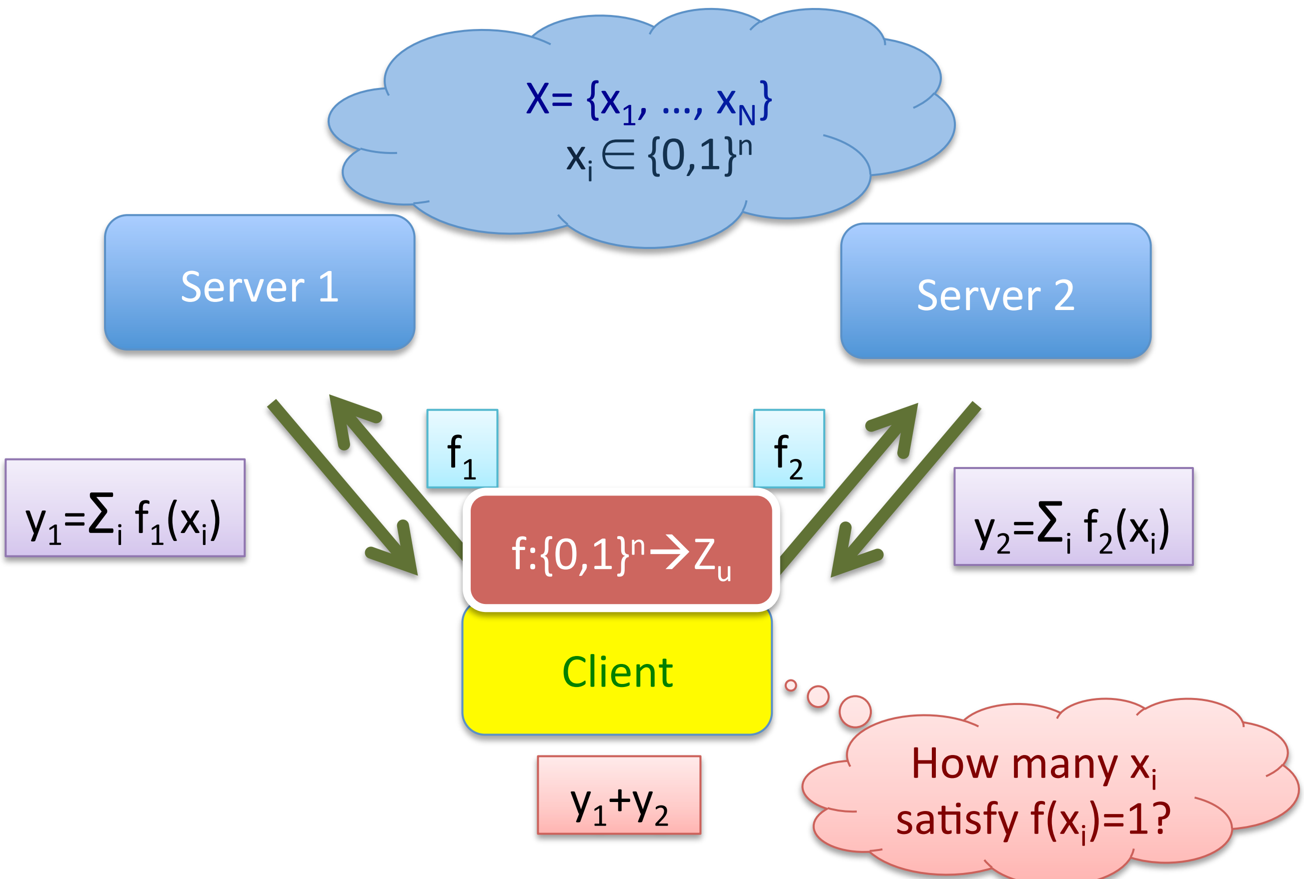
Applications: Reading

- Keyword search with payloads



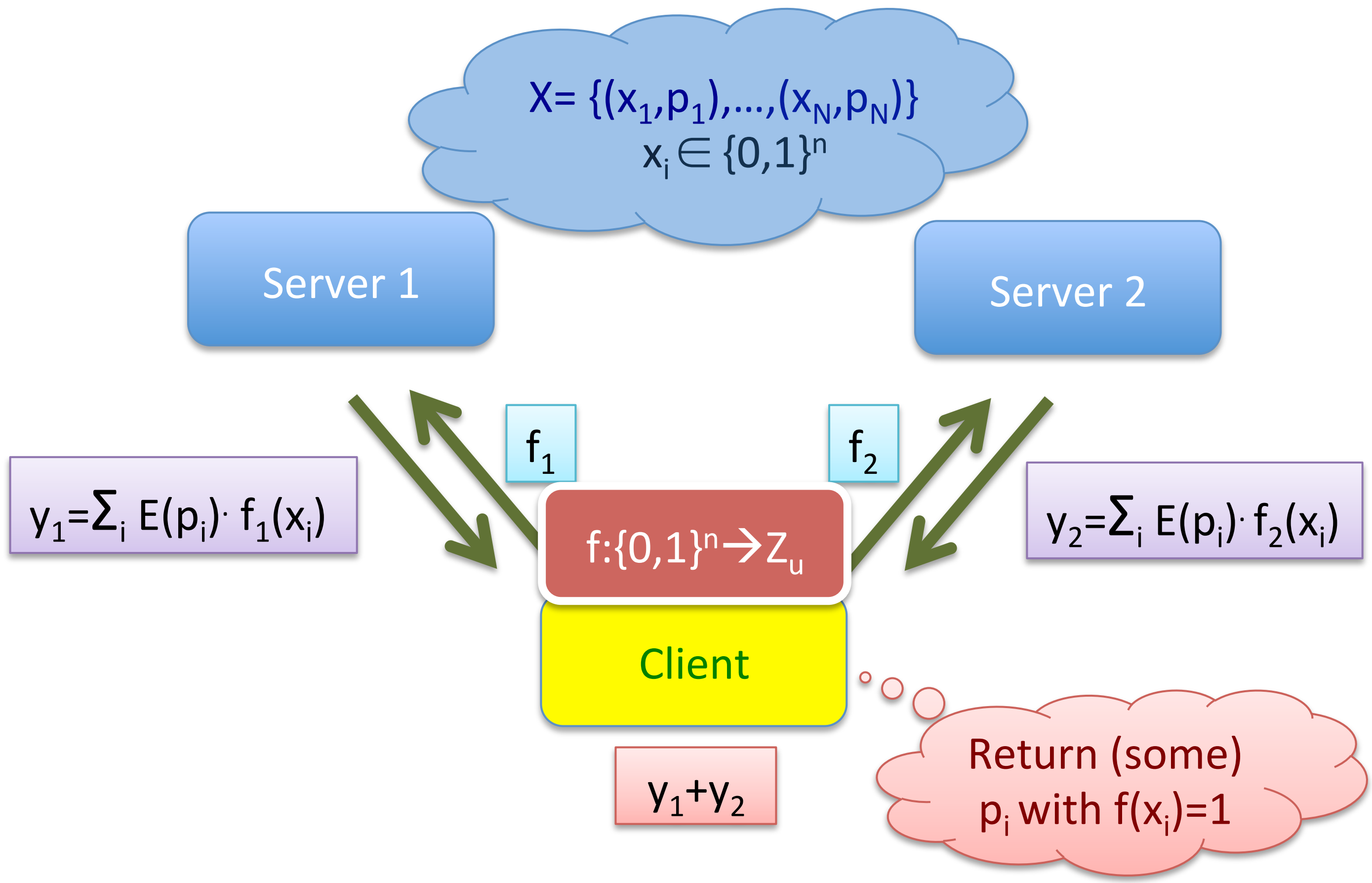
Applications: Reading

- Generalized keyword search



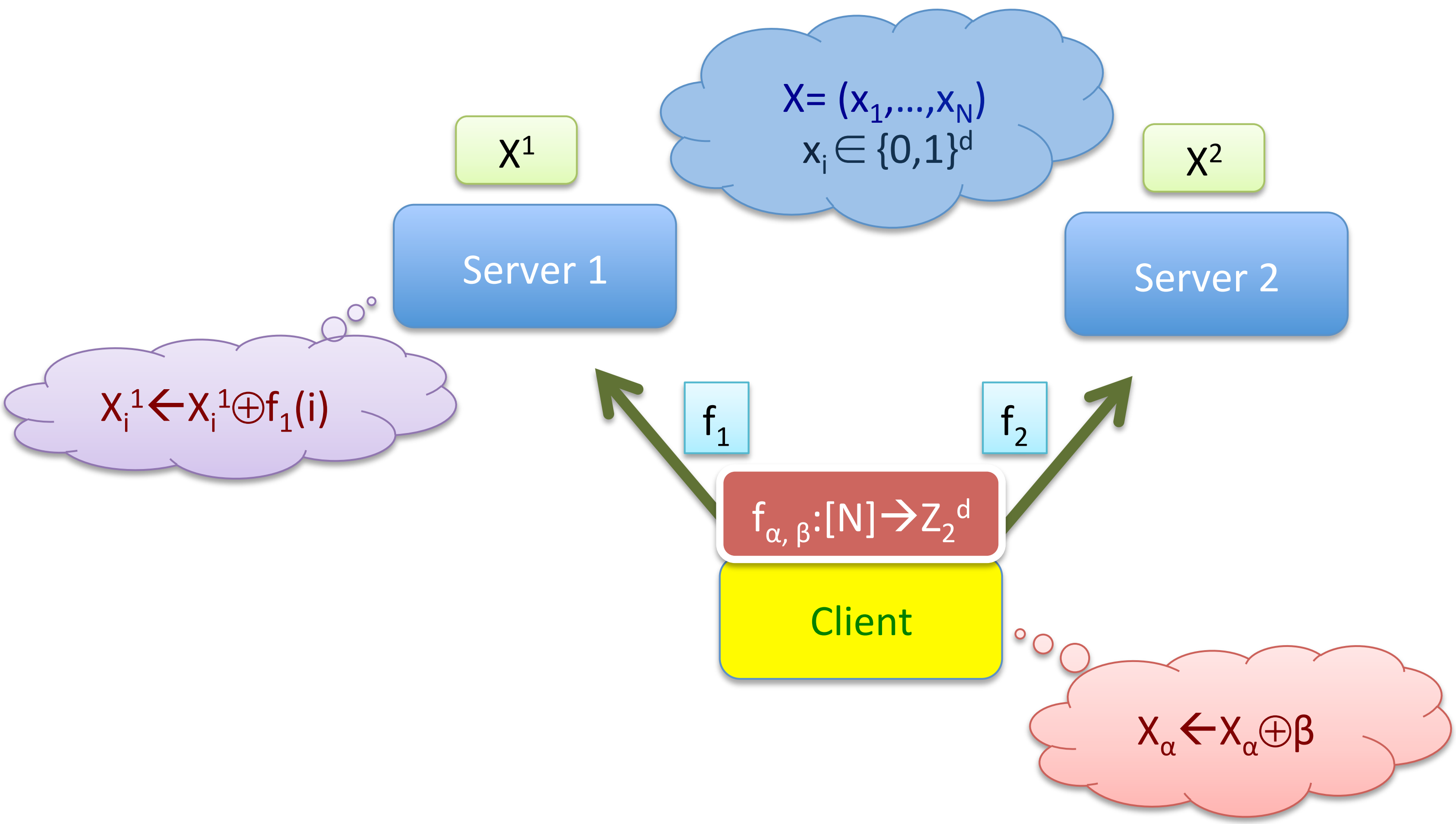
Applications: Reading

- Generalized keyword search with payloads?



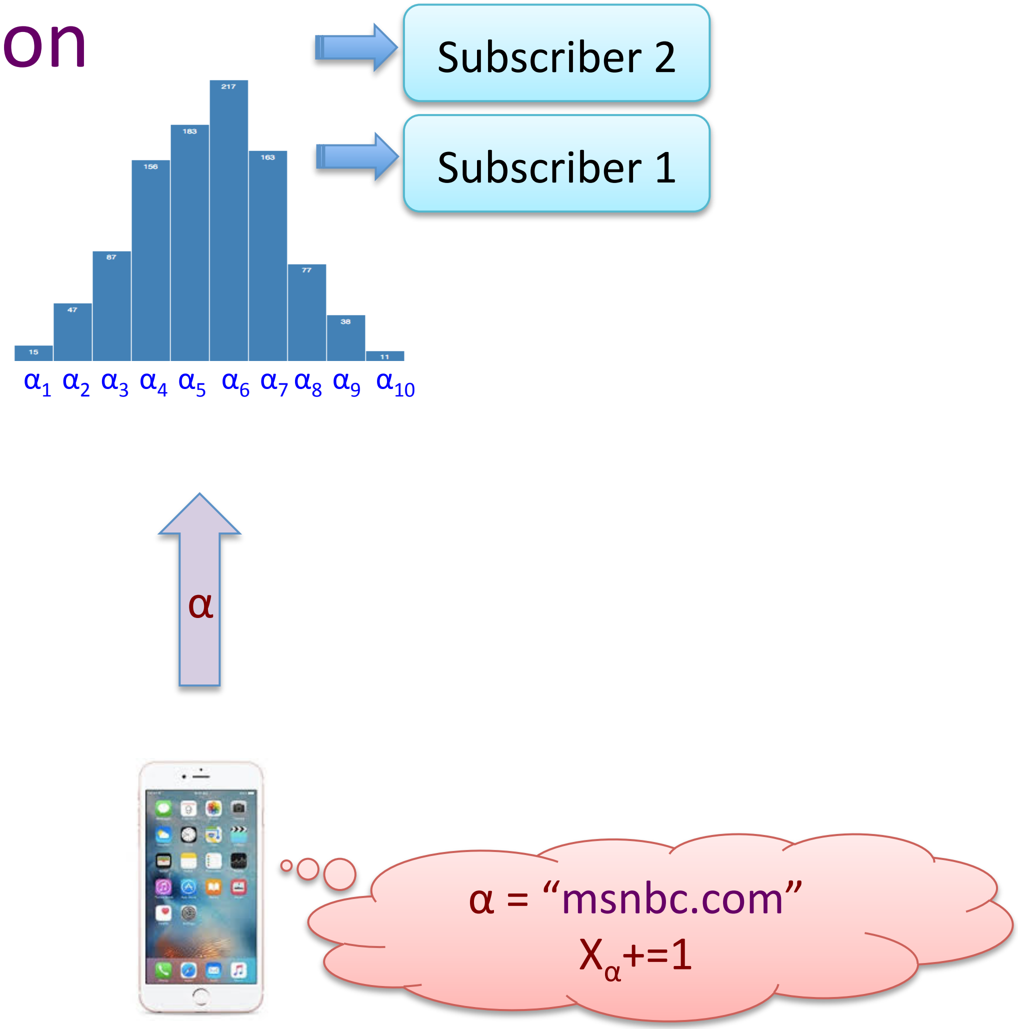
Applications: Writing

- PIR-writing [OS98,…] ("private information storage")



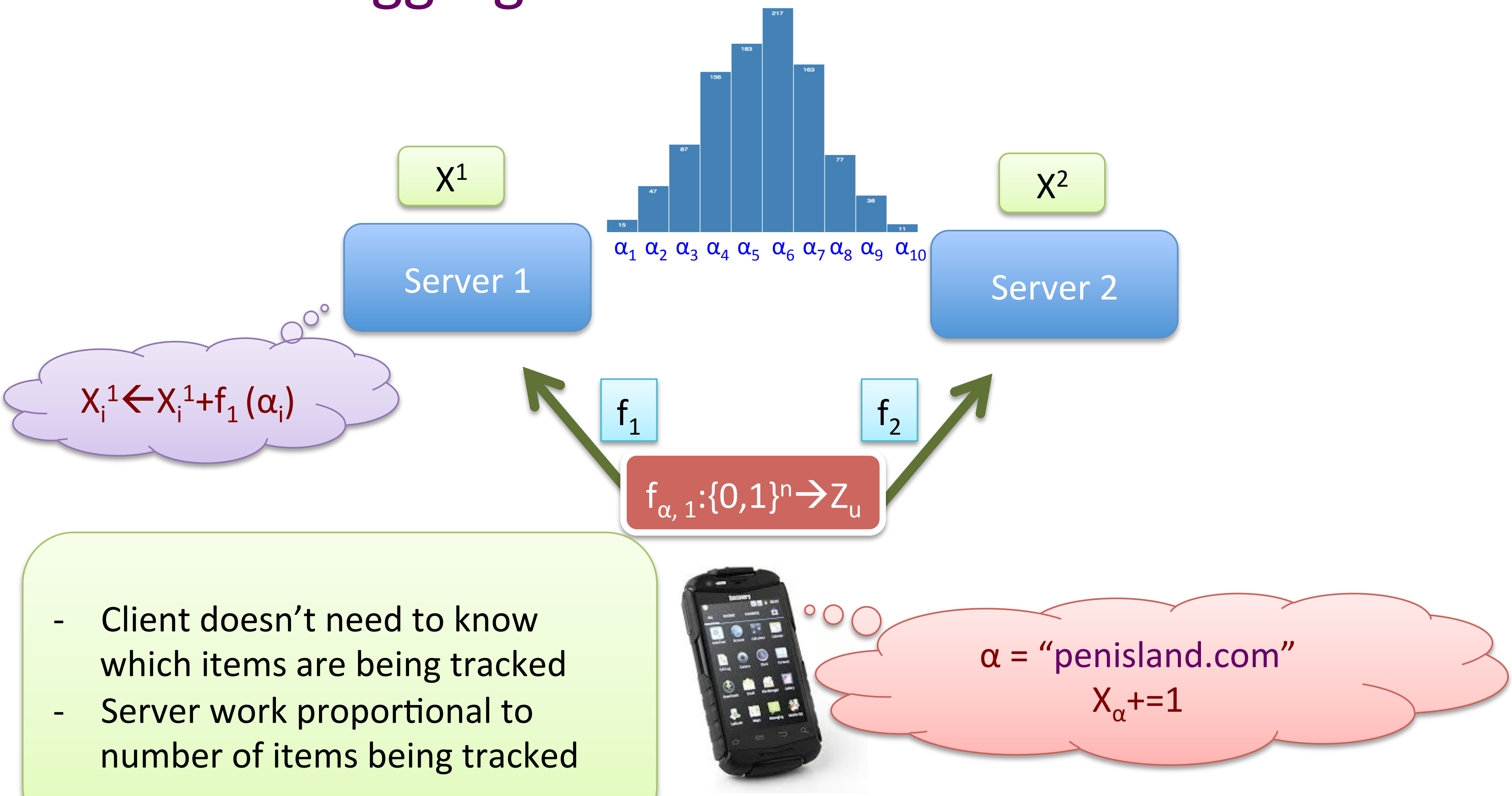
Applications: Writing

- Secure aggregation



Applications: Writing

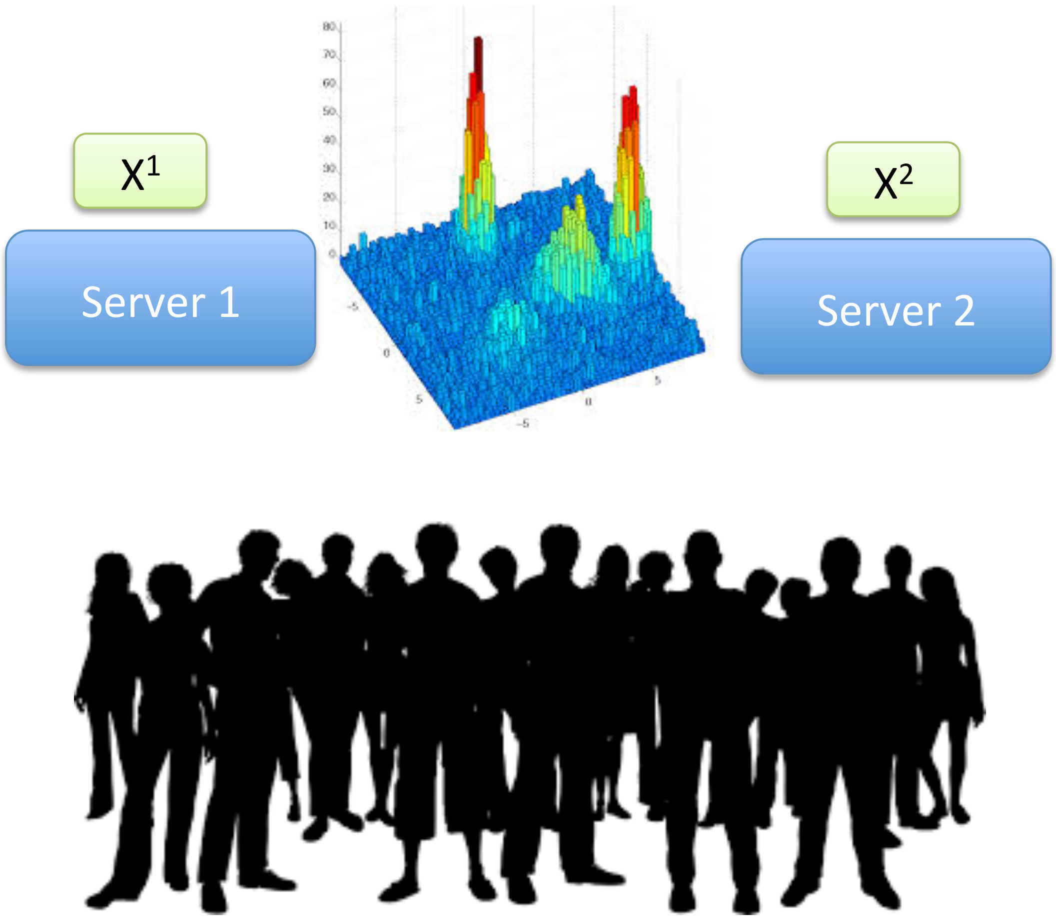
- Secure aggregation



- Client doesn’t need to know which items are being tracked
- Server work proportional to number of items being tracked

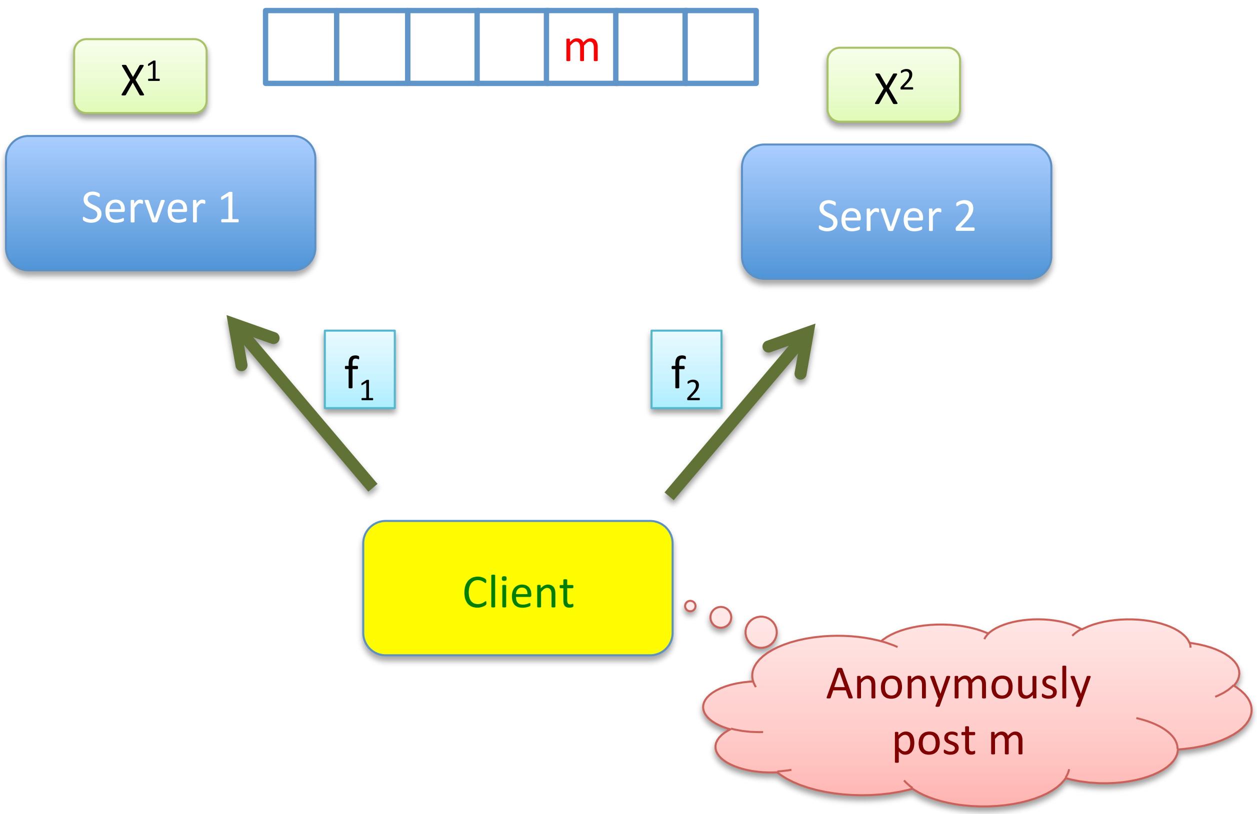
Applications: Writing

- Large scale MPC over small domains



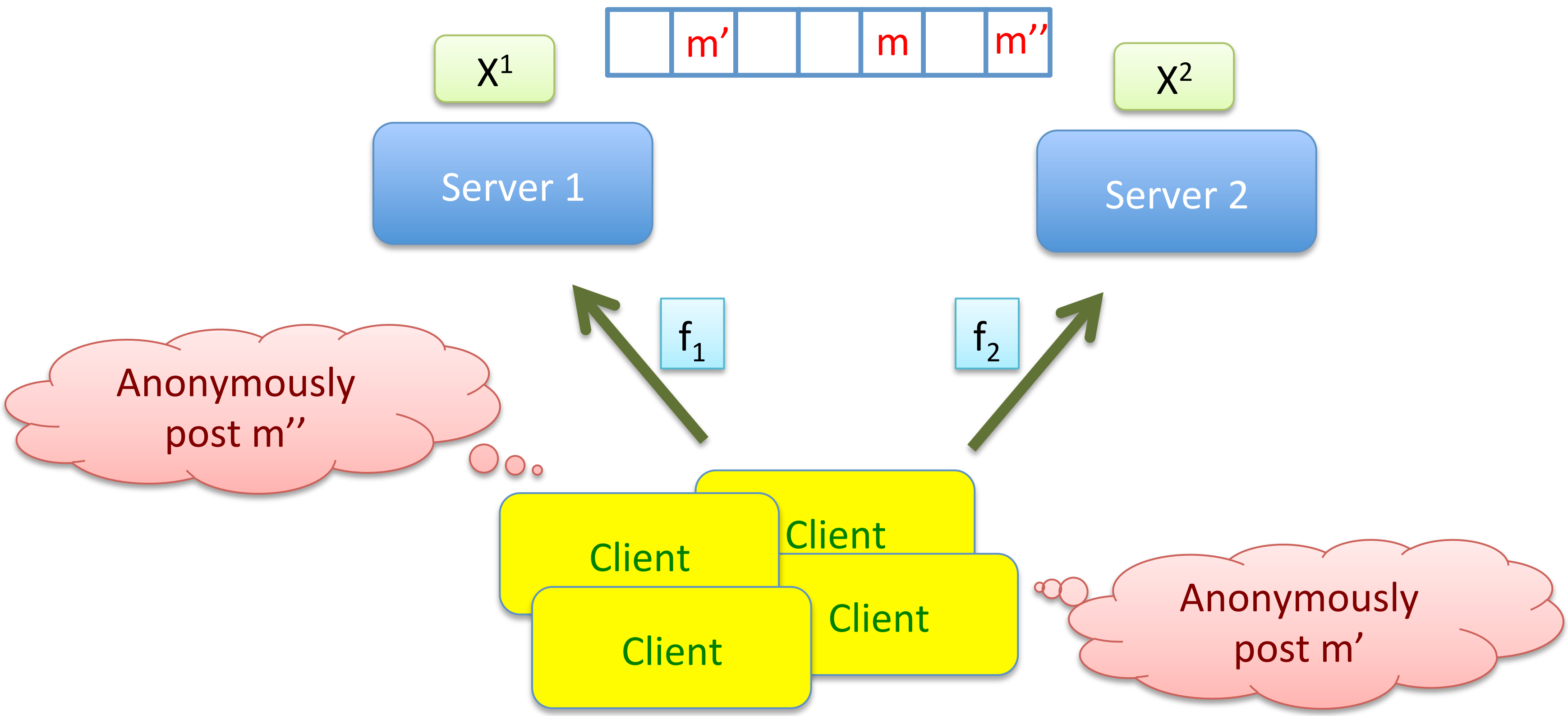
Applications: Writing

- Anonymous messaging [CBM15]



Applications: Writing

- Anonymous messaging [CBM15]

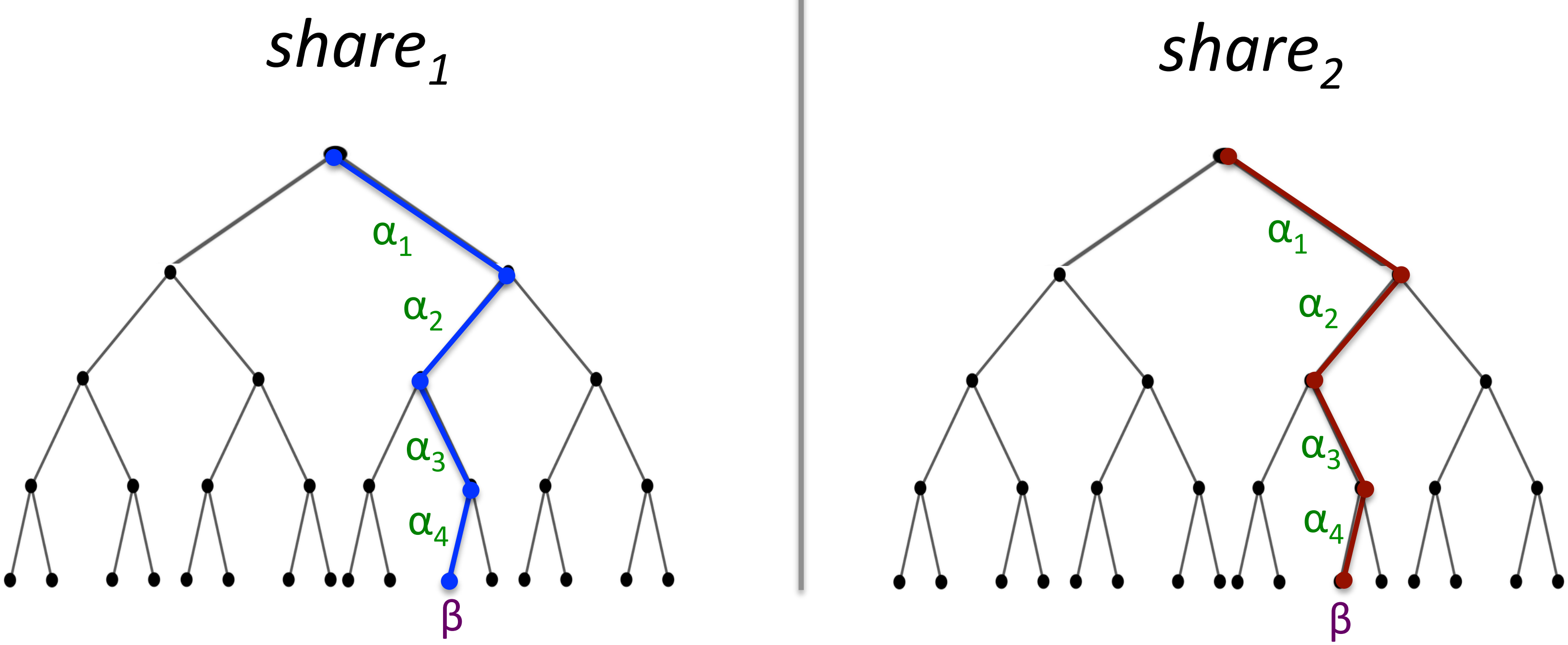


PRG-based DPF

- Let <x> denote additive (XOR) secret sharing
  - $<x>=(x_1,x_2)$ s.t. $x_1-x_2=x$

- Exploit two simple types of homomorphism
  - Additive: <x> , <y> → <x+y> by local addition
  - Weak expansion: <x> → <X> by locally applying PRG
    - $x=0^{\lambda}$ → $X=0^{2\lambda}$
    - x = random → X = pseudo-random

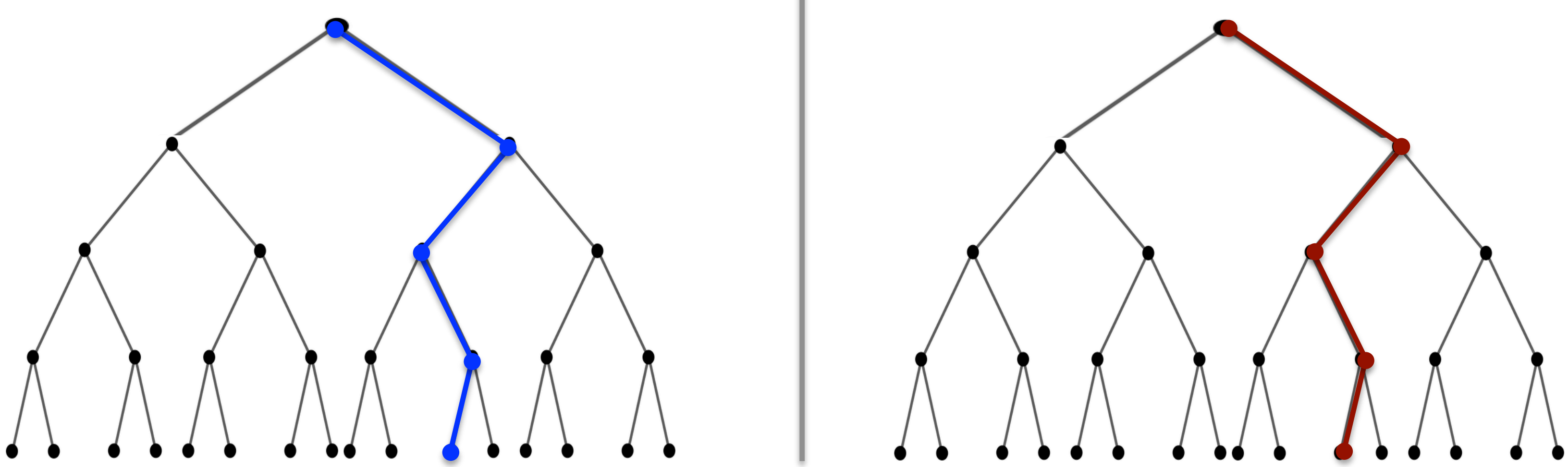
PRG-based DPF



Shares define two correlated "GGM-like" trees

PRG-based DPF

share$_1$ *share*$_2$

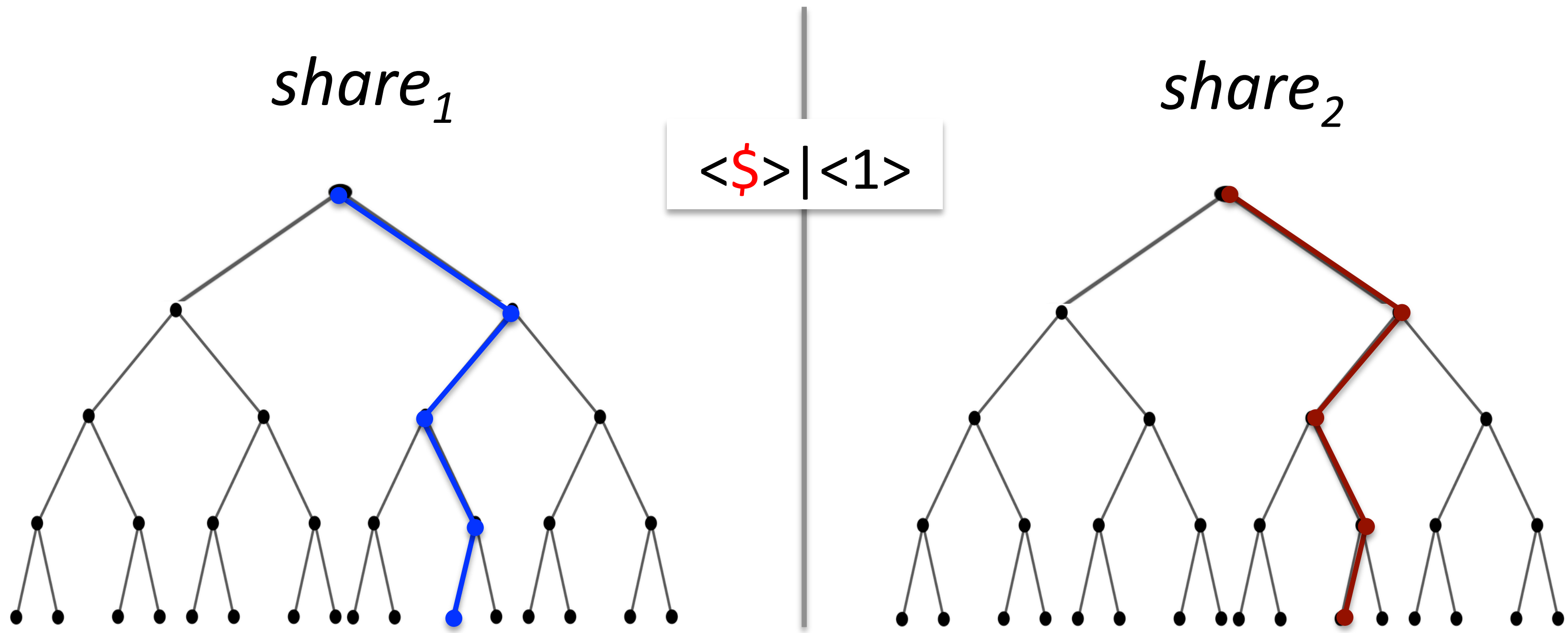


Invariant for Eval:

λ-bit 1-bit

For each node v on evaluation path we have <S>|<b>

PRG-based DPF

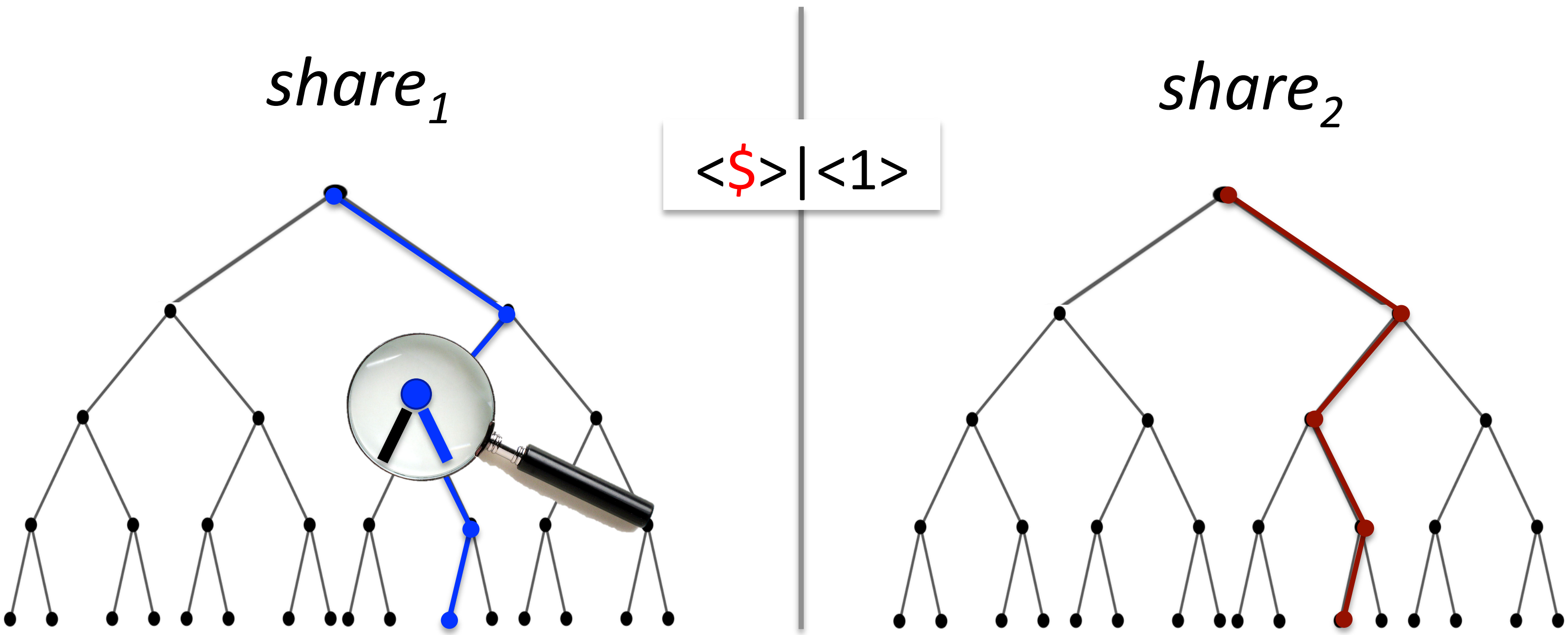


Invariant for Eval:

For each node v on evaluation path we have <S>|<b>

- v on special path: S is pseudorandom, b=1
- v off special path: S=0, b=0

PRG-based DPF



Invariant for Eval:

For each node v on evaluation path we have <S>|<b>

- v on special path: S is pseudorandom, b=1
- v off special path: S=0, b=0

Gadget: Conditional Correction

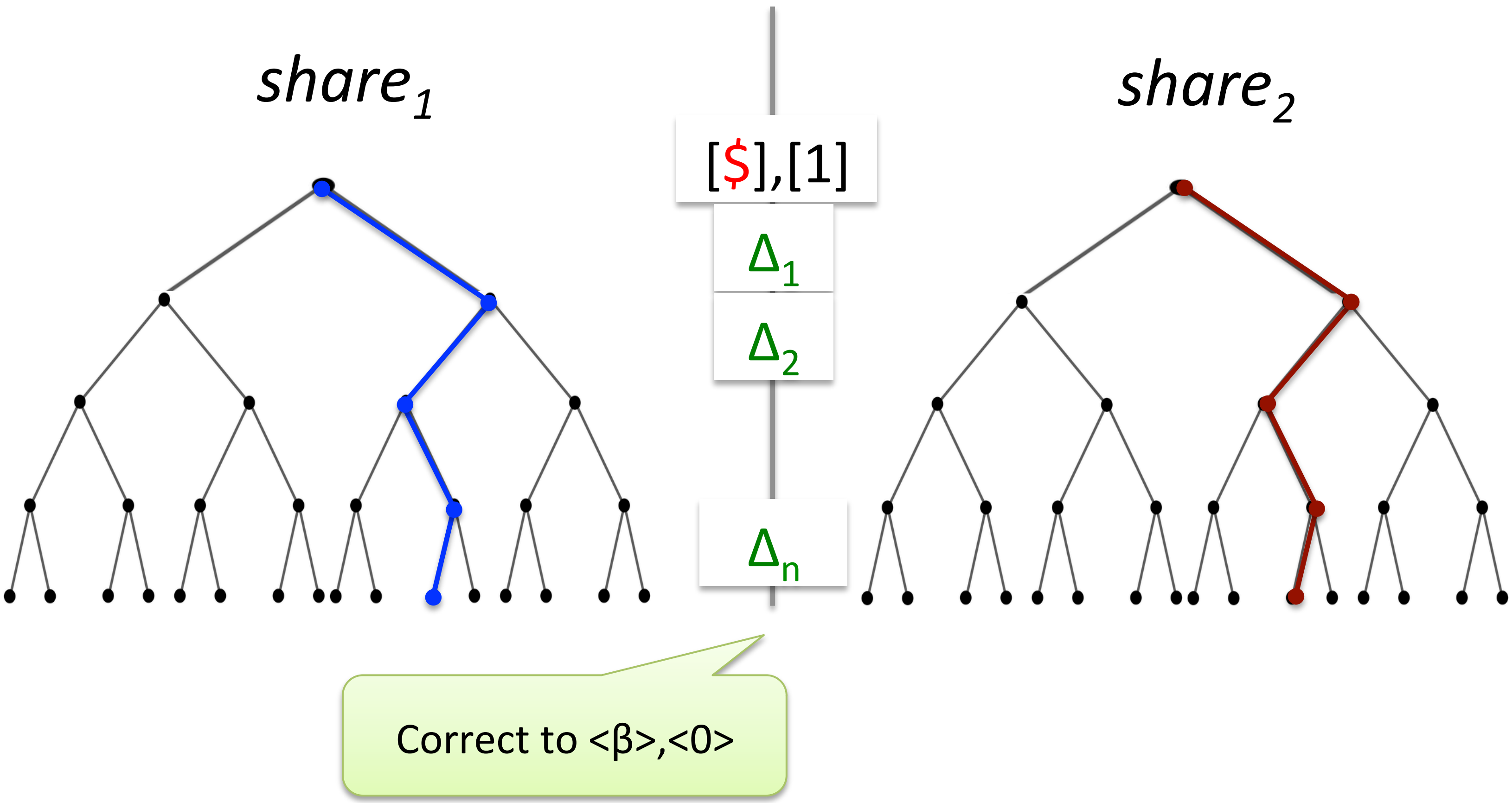
$R_1 \in \{0,1\}^k$ $<R>$ $R_2 = R_1 \oplus R$

$b_1 \in \{0,1\}$ $<b>$ $b_2 = b_1 \oplus b$

$\Delta \in \{0,1\}^k$

$R_1 \oplus b_1 \cdot \Delta$ $<R \oplus b \cdot \Delta>$ $R_2 \oplus b_2 \cdot \Delta$

PRG-based DPF



Concrete Efficiency of DPF

- Share size $\cong n \cdot \lambda$, for PRG: $\{0,1\}^\lambda \rightarrow \{0,1\}^{2(\lambda+1)}$
  - Slightly better for binary output

- Concrete cost of Eval $\cong$ n x PRG, Gen $\cong$ 2 x Eval
  - Evaluating on the entire domain [N] $\cong N/\lambda$ x PRG (N/64 x AES)

- Example: 2-server PIR on 2^{25} records of length d
  - Communication: 2578 bits to each server, d bits in return
  - Computation: dominated by reading + XORing all records

Extensions

- m-party DPF from PRG [BGI15]
  - Near-quadratic improvement over naive solution
    … with 2^m overhead

- FSS for intervals, decision trees (leaking topology), d-dimensional intervals [BGI16]

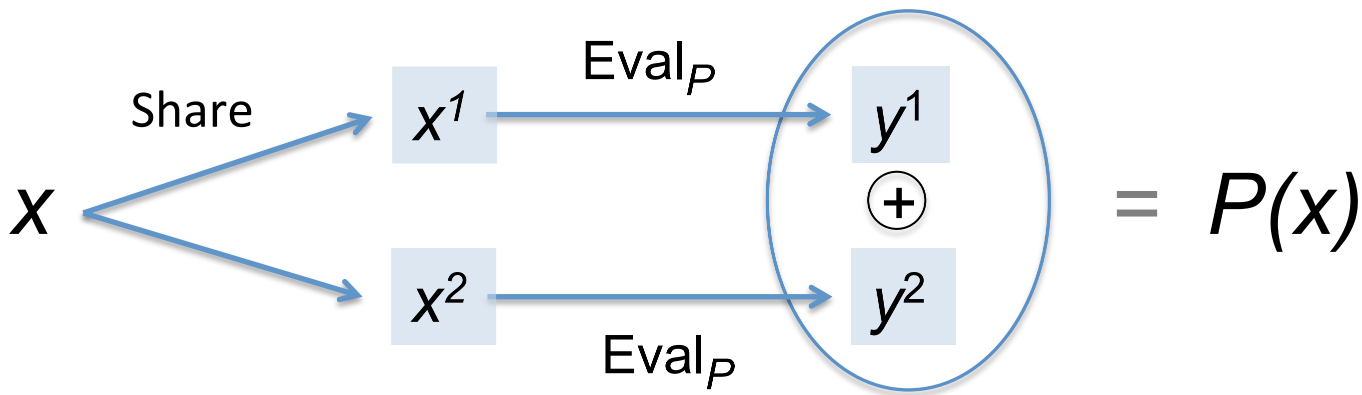
- Barrier (?): FSS for class F containing decryption → Succinct 2PC for F from OT (w/reusable preprocessing)
  - Meaningful even for F=AC^0
  - May lead to positive results!

Open Problems: FSS from OWF

- 3-party DPF
  - $o(N^{1/2})$ key size from OWF?
- Limits of 2-party FSS from OWF
  - FSS for conjunctions / partial match?
  - Stronger barriers
- Power of information-theoretic (m,t)-FSS
  - Even 2-party FSS with non-additive output
- Efficiency of 2-party DPF
  - Beat $n \cdot \lambda$ key size?
  - Amortizing cost of multi-point DPF?

HSS for Branching Programs from DDH

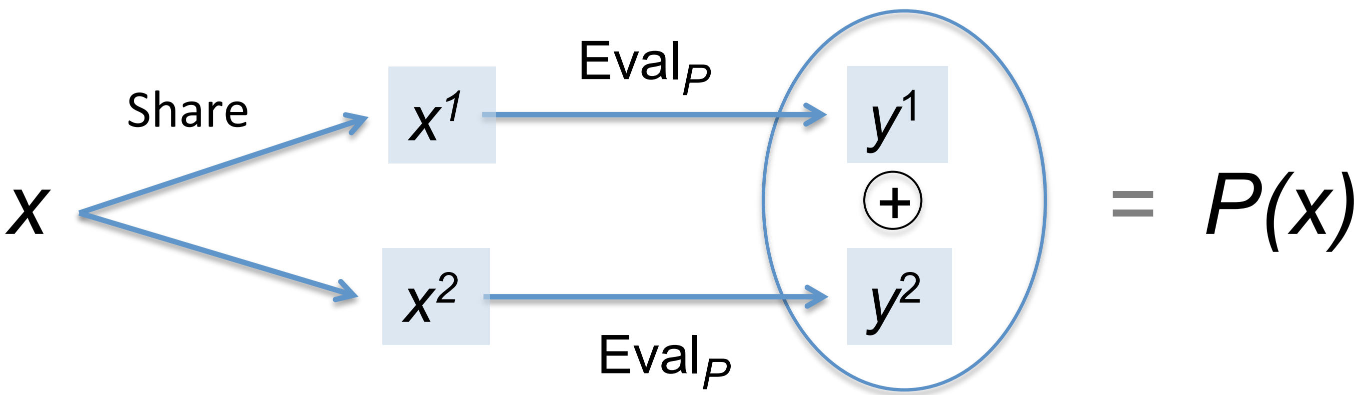
Recall: Homomorphic Secret Sharing



- Security: x^i hides x
- Correctness:

$$\mathrm{Eval}_P(x^1) + \mathrm{Eval}_P(x^2) = P(x)$$

δ-HSS



- **Security:** x^i hides x
- **δ-Correctness:** Except with prob. δ (over Share),

$$\mathrm{Eval}_P(x^1) + \mathrm{Eval}_P(x^2) = P(x)$$

Main Theorem

- 2-party δ-HSS for branching programs under DDH
  - Share: runtime (& share size) = $|x| \cdot \text{poly}(\lambda)$
  - Eval: runtime = $\text{poly}(\lambda, |P|, 1/\delta)$
    for error probability δ

Living in a log-space world

Multiplication of *n* *n*-bit numbers

Streaming algorithms

Min L_2-distance from list of length-*n* vectors

Many numerical / statistical calculations

Finite automata

Undirected graph connectivity

FHE Decryption

…

The HSS Construction

RMS Programs

Restricted-Multiplication Straight-line programs:

- $v_i \leftarrow x_j$ Load an input into memory.
- $v_i \leftarrow v_j + v_k$ Add values in memory.
- $v_i \leftarrow v_j * x_k$ Multiply value in memory by an *input*.
- Output v_i (mod m)

We will support homomorphic evaluation of RMS programs over Z s.t. all intermediate values are "small" (e.g., {0,1})

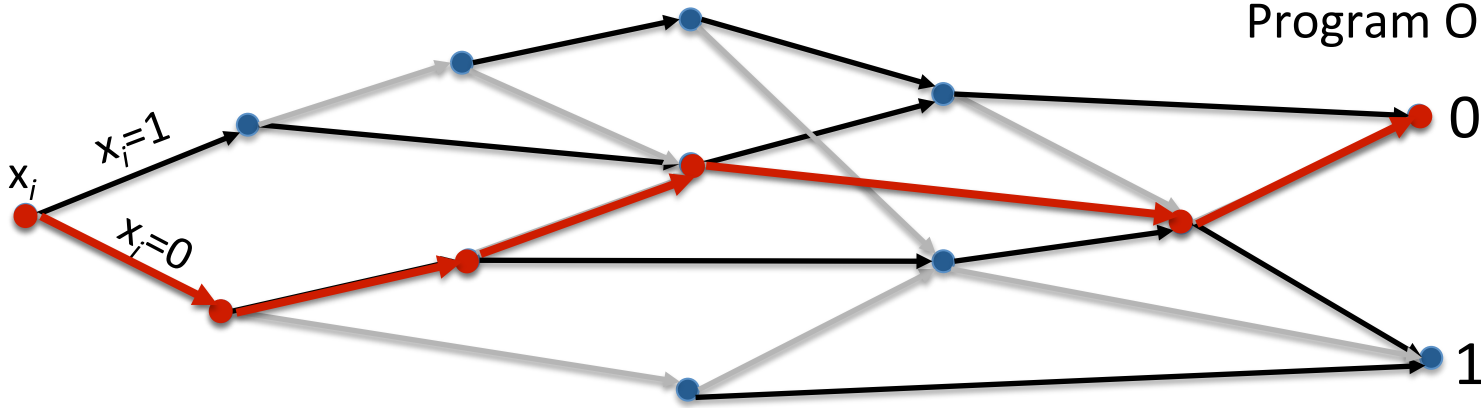
Captures branching programs and log-space computations

(More generally: ReachFewL)

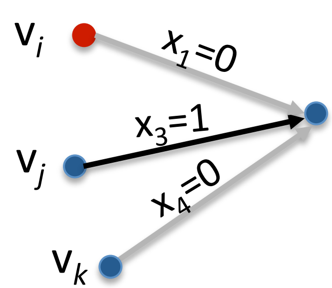
RMS Captures Branching Programs

Program Input: $x_1\ x_2\ x_3\ x_4 \ldots x_n$

Program Output:



To evaluate as RMS: Memory variable for each *node* (whether it's on red path)



$v_\ell = (1-x_1)\, v_i + (x_3)\, v_j + (1-x_1)\, v_k$

Computable via RMS

3 Ways to Share a Number

- Let G be a DDH group of size q with generator g
- 3 levels of encoding Z_q elements
  - [u] : $(g^u, g^u) \in G \times G$ "encryption"
  - <v> : $(v_1,v_2) \in Z_q \times Z_q$ s.t. $v_1=v_2+v$ additive
  - {w} : $(w_1,w_2) \in G \times G$ s.t. $w_1=w_2 \cdot g^w$ multiplicative
- Each level is additively homomorphic
  - <v>,<v'>➔<v+v'> {w},{w'}➔{w+w'}
- Natural pairing: pair([u],<v>) ➔ {uv}
  - $((g^u)^\wedge v_1,(g^u)^\wedge v_2)=(g^{uv2}\cdot g^{uv},g^{uv2})$

Toy Version

Let's pretend g^x is a secure encryption of x

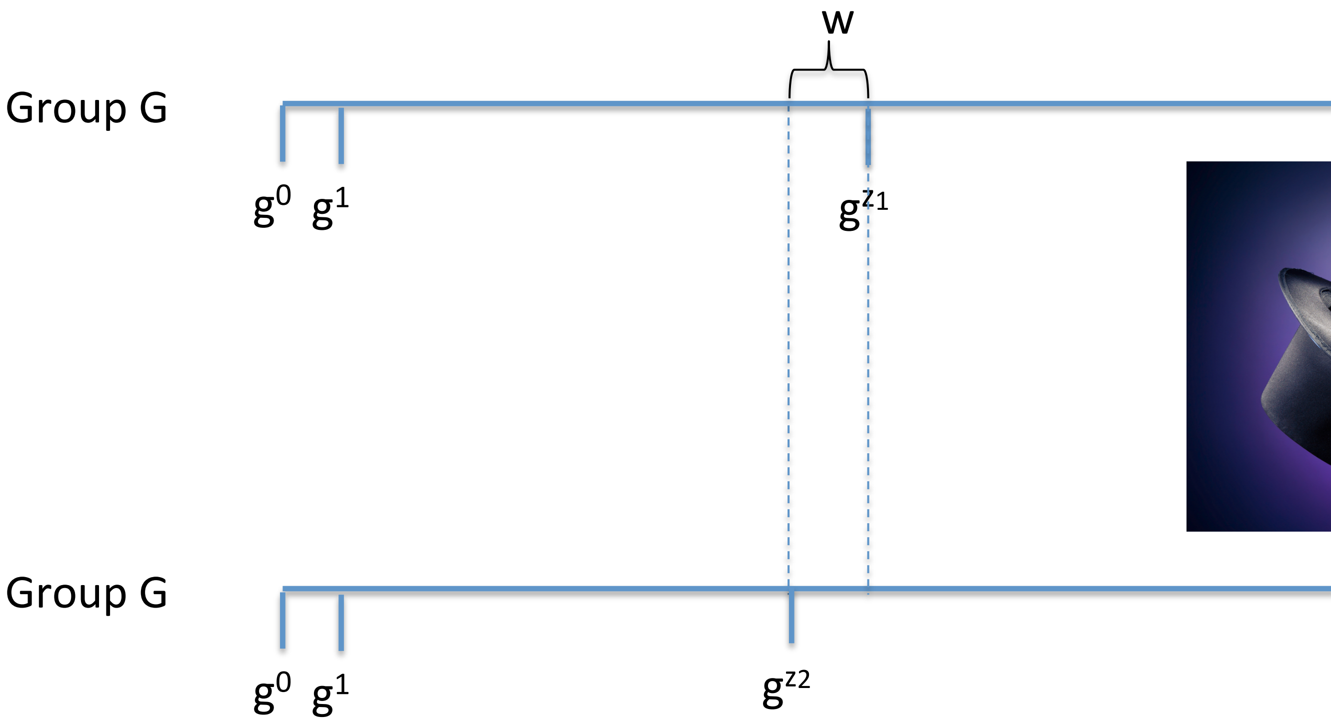
$[u]=(g^u,g^u)$
$\langle v\rangle=(v_2+v,v_2)$
$\{w\}=(w_2\cdot g^w,w_2)$

Emulating an RMS program – first attempt:

- Share: for each input x_i
  - Encrypt as $[x_i]$
  - Additively secret-share as $\langle x_i\rangle$
- Eval: // maintain the invariant: $V_i = \langle v_i\rangle$
  - $v_i \leftarrow x_j$: $V_i \leftarrow \langle x_j\rangle$
  - $v_i \leftarrow v_j+v_k$: $V_i \leftarrow V_j+V_k$ // $V_i=\langle v_j+v_k\rangle$
  - Output v_i (mod m): Output V_i +(r,r) (mod m)
  - $v_i \leftarrow x_k{*}v_j$: $W_i \leftarrow \text{pair}([x_k],V_j)$ // $W_i=\{w\}$ for $w=x_k\cdot v_j$

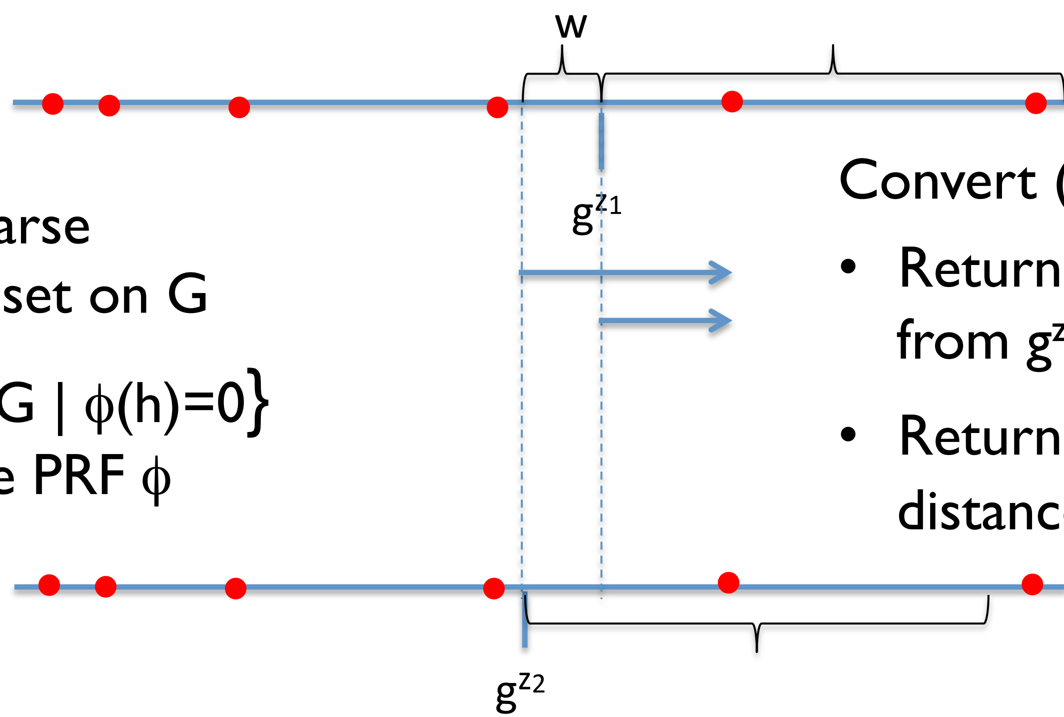
Need Convert : $\{w\} \rightarrow \langle w\rangle$
Solved by discrete log…
Stuck?

Share Conversion



Goal: Locally convert multiplicative sharing of w to additive sharing of w

Share Conversion



S is a δ-sparse "random" set on G

eg $S=\{h\in G \mid \phi(h)=0\}$ for suitable PRF ϕ

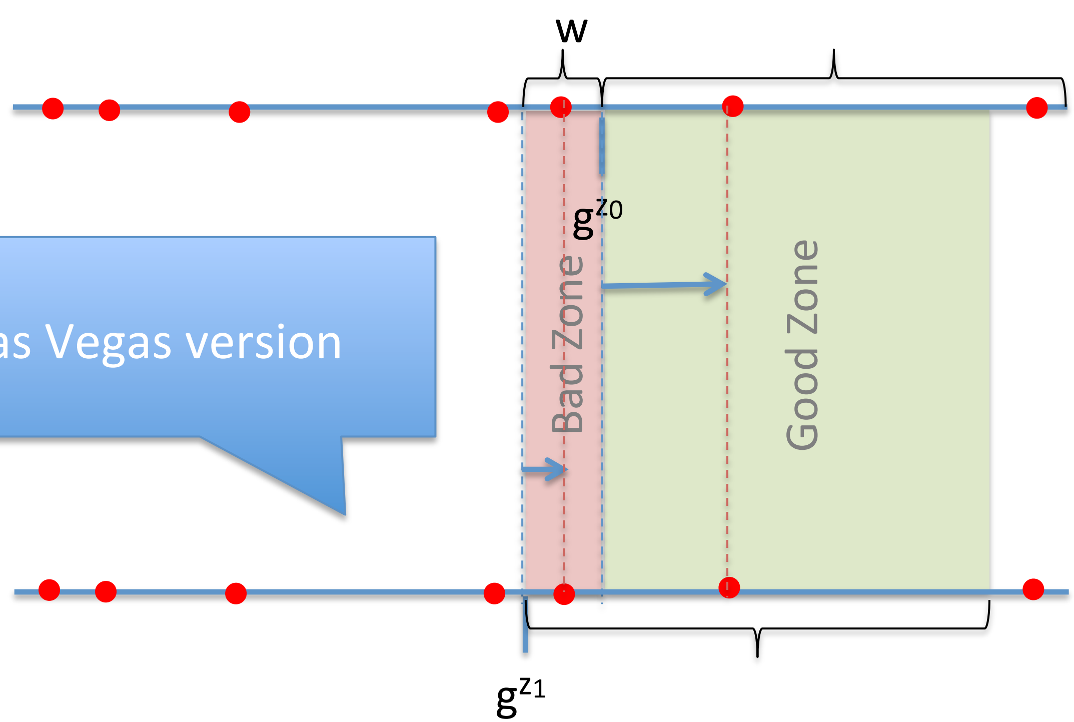
Convert (g^{z_b}):

- Return distance $dist_b$ from g^{z_b} to S.
- Return $dist_b=0$ if distance$>(1/\delta)\cdot\log(1/\delta)$

Goal: Convert multiplicative sharing of w to additive sharing of w

Conversion Error

Las Vegas version



Bad cases:

$\exists$ • $\in$ Bad Zone   error ~ δw

$\nexists$ • $\in$ Good Zone   error ~ δ

Error probability depends on w

Toy Version

Let's pretend g^x is a secure encryption of x

$[u]=(g^u, g^u)$
$\langle v\rangle=(v_2+v, v_2)$
$\{w\}=(w_2 \cdot g^w, w_2)$

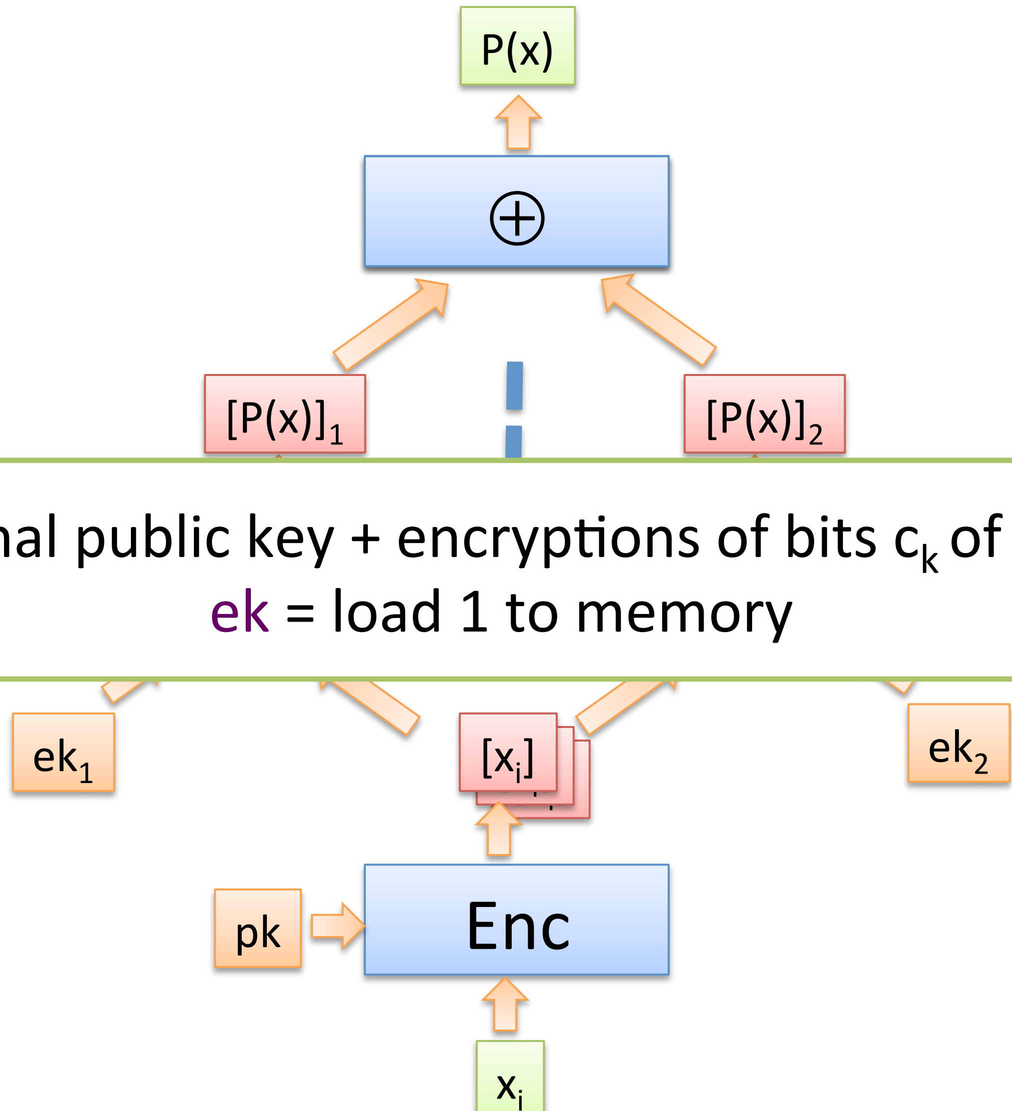
Emulating an RMS program:

- Share: for each input x_i
  - Encrypt as $[x_i]$
  - Additively secret-share as $\langle x_i \rangle$
- Eval: // maintain the invariant: $V_i = \langle v_i \rangle$
- $v_i \leftarrow x_j$: $V_i \leftarrow \langle x_j \rangle$
- $v_i \leftarrow v_j+v_k$: $V_i \leftarrow V_j+V_k$ // $V_i = \langle v_j+v_k \rangle$
- $v_i \leftarrow x_k * v_j$: $W_i \leftarrow \text{pair}([x_k],V_j)$; $V_i \leftarrow \text{Convert}(W_i)$
- Output v_i (mod m): Output V_i mod m

From Toy Version to Real Version

- Pick secret key $c \in Z_q$ for ElGamal encryption
- Encrypt each input x_i as $[r], [cr+x_i]$ (secret-key ElGamal)
- Invariant: Each memory value v_j shared as $\langle v_j \rangle, \langle cv_j \rangle$
- To multiply $x_i v_j$: pair, subtract and get $\{x_i v_j\}$
  - Use conversion to get $\langle x_i v_j \rangle$
  - Problem: Need also $\langle c \cdot x_i v_j \rangle$ to maintain invariant
  - Solution? Share $c \cdot x_i$ in addition to x_i
  - Problem: Can't convert $\{c \cdot x_i v_j\}$ ($c \cdot x_i v_j$ too big)
  - Solution: Break c into binary representation, encrypt $x_i c_k$
  - Problem: circular security for ElGamal?
  - Solutions: (1) assume it! (2) leveled version (3) use [BHHO08]

Public-Key Variant



pk = ElGamal public key + encryptions of bits c_k of secret key

ek = load 1 to memory

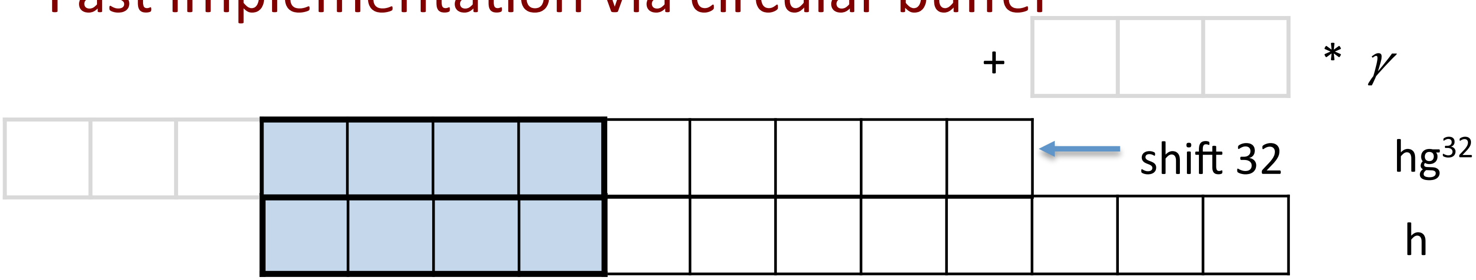
Applications

- Succinct 2PC for branching programs / logspace / NC^1
  - Communication |inputs| + |outputs| + poly(λ) **bits**
- Sublinear 2PC for "nice" circuits
  - Communication O(|C|/log|C|) + … **bits**
  - O(|C|)+… bits for general circuits
- 2-server PIR for branching program queries
- 2-party FSS for branching programs
- 2-round MPC in PKI model
  - O(1) parties

Computational Optimizations

- "Conversion-friendly" groups:
  $g = 2$ is generator & $p = 2^i$ - (small)
  $h \cdot g$ = (shift 1) + small

- Distinguished points:
  - Index of minimum value of min-wise hash
    Saves $\log(1/\delta)$ factor in worst-case runtime
  - Heuristic: sequence 0^d
    Fast implementation via circular buffer



Further Optimizations

- Assume circular-secure ElGamal
- Elliptic-curve ElGamal for short ciphertexts
- “Small exponent” ElGamal for shorter secret key
- Preprocess for fixed-basis exponentiations
- Replace binary sk decomposition by base D

- Bottom line:
  - Orders of magnitude improvement compared to baseline
  - Ciphertexts and keys shorter than in FHE
  - Fast enough for non-trivial applications [BCGIO17]

Conclusions

- Homomorphic secret sharing from DDH
  - Supports branching program computation
  - Yields succinct secure computation and other applications of FHE
  - Some applications not implied by standard FHE
  - Good concrete efficiency for "shallow" computations
- Not post-quantum
  - I have bigger concerns at this moment
  - Quantum-friendly cryptography?

Open Questions

- Beyond branching programs
  - FHE-style bootstrapping?
- More than 2 parties
- Different assumptions
  - Paillier [Gennaro-Jafarikah-Skeith17, Couteau17]
  - QRA? LPN? Better from LWE?
- Better time/error tradeoff of conversion?
- Fault tolerance at branching program level?
- Better concrete efficiency