Homomorphic Secret Sharing

Elette Boyle  Niv Gilboa  Yuval Ishai
IDC  BGU  Technion & UCLA
1970
1980
1990
2000
2010

Primitives

PKE
Signatures
ZK
OT

Secure Computation

Assumptions

Factoring
Discrete Log
1970
1980
1990
2000
2010

Primitives

PKE
Signatures ZK OT
Secure Computation

Assumptions

Factoring Discrete Log

• Minimize communication?
• Minimize interaction?
• Minimize local computation?
Primitives

- PKE
- Signatures
- ZK
- OT
- Secure Computation

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]

• 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

Given a generic group G:
  • Unconditionally secure PKE and even secure computation
  • Not known to be helpful for FHE
IN SOME SENSE

THERE HAS GOT TO BE A BETTER WAY
Recall: FHE

\[
P(x) \xrightarrow{sk} \text{Dec} \xrightarrow{[P(x)]} \text{Eval} \xrightarrow{[x]} \text{Enc} \xrightarrow{pk} x
\]
“1/2 FHE”

Dec

Enc

P(x)

sk

[P(x)]_1

[x]_1

pk

[P(x)]_2

[x]_2

computationally hides x

computationally hides x
(2-Party) Homomorphic Secret Sharing
(2-Party) Homomorphic Secret Sharing

P(x)

+  

[P(x)]_1  [P(x)]_2

Eval

[x]_1  [x]_2

Eval

Share

P
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

FHE

HSS

\[ \text{sk} \rightarrow [x] \rightarrow [P(x)] \rightarrow \text{P(x)} \]

\[ [x]_1 \rightarrow [P(x)]_1 \rightarrow \oplus \rightarrow \text{P(x)} \]
Applications

Delegating Computations to the Cloud

Bonus features:
- Multiple clients
- Useful also for small $P$
Applications

Communication complexity of securely computing $C$?

• Classically: $> |C| \quad [\text{Yao86,GMW87,BGW88,CCD88,}]$
  ... even for restricted classes, such as formulas

• Using FHE: $\sim |\text{input}| + |\text{output}|$
Applications

Succinct Secure Computation

FHE

HSS

Bonuses features:
- Beats FHE for long outputs
- Useful for generating correlations
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 \( f_{\alpha,\beta} : \{0,1\}^n \to G \)
  
  \( f_{\alpha,\beta}(\alpha) = \beta \)
  
  \( f_{\alpha,\beta}(x) = 0 \) 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, ...]

\[ X = \{ x_1, \ldots, x_N \} \]
\[ x_i \in \{0,1\}^n \]

\[ y_1 = \bigoplus_i f_1(x_i) \]
\[ y_2 = \bigoplus_i f_2(x_i) \]

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

• Keyword search with payloads

\[ X = \{(x_1, p_1), ..., (x_N, p_N)\} \]
\[ x_i \in \{0,1\}^n \]

\[ y_1 = \bigoplus_i p_i f_1(x_i) \]
\[ f_{x,1}: \{0,1\}^n \rightarrow \mathbb{Z}_2 \]
\[ y_2 = \bigoplus_i p_i f_2(x_i) \]

Client

\[ y_1 \oplus y_2 \]

Get payload of keyword x
Applications: Reading

• Generalized keyword search

\[ X = \{x_1, \ldots, x_N\} \]
\[ x_i \in \{0,1\}^n \]

Server 1

\[ y_1 = \sum_i f_1(x_i) \]

Server 2

\[ y_2 = \sum_i f_2(x_i) \]

\[ f : \{0,1\}^n \rightarrow \mathbb{Z}_u \]

Client

\[ y_1 + y_2 \]

How many \( x_i \) satisfy \( f(x_i) = 1? \)
Applications: Reading

- Generalized keyword search with payloads?

\[ X = \{(x_1, p_1), \ldots, (x_N, p_N)\} \]
\[ x_i \in \{0,1\}^n \]

Server 1

\[ y_1 = \sum_i E(p_i) \cdot f_1(x_i) \]

Server 2

\[ y_2 = \sum_i E(p_i) \cdot f_2(x_i) \]

Client

\[ f : \{0,1\}^n \rightarrow \mathbb{Z}_u \]

\[ y_1 + y_2 \]

Return (some) \( p_i \) with \( f(x_i) = 1 \)
Applications: Writing

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

\[ X = (x_1, \ldots, x_N) \quad x_i \in \{0,1\}^d \]

\[ f_1, f_2 : [N] \rightarrow \mathbb{Z}_2^d \]

\[ x_i^1 \leftarrow x_i^1 \oplus f_1(i) \]

\[ f_{\alpha, \beta} : [N] \rightarrow \mathbb{Z}_2^d \]

\[ x_\alpha \leftarrow x_\alpha \oplus \beta \]
Applications: Writing

• Secure aggregation

\[ \alpha \text{ "msnbc.com" } X_\alpha \geq 1 \]
Applications: Writing

- Secure aggregation

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

\( \alpha = \text{“penisland.com”} \)

\( X_\alpha +=1 \)
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: \(<x>,<y>\rightarrow<x+y>\) by local addition
  – Weak expansion: \(<x>\rightarrow<X>\) by locally applying PRG
    • \(x=0^\lambda\rightarrow X=0^{2\lambda}\)
    • \(x=\text{random}\rightarrow X=\text{pseudo-random}\)
PRG-based DPF

 Shares define two correlated “GGM-like” trees
PRG-based DPF

Invariant for Eval:

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: $S=0$, $b=0$
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

\[ R_1 \in \{0,1\}^k \quad \langle R \rangle \quad R_2 = R_1 \oplus R \]

\[ b_1 \in \{0,1\} \quad \langle b \rangle \quad b_2 = b_1 \oplus b \]

\[ \Delta \in \{0,1\}^k \]

\[ R_1 \oplus b_1 \cdot \Delta \quad \langle R \oplus b \cdot \Delta \rangle \quad R_2 \oplus b_2 \cdot \Delta \]
PRG-based DPF

Correct to \( \langle \beta \rangle, \langle 0 \rangle \)
Concrete Efficiency of DPF

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

• Concrete cost of Eval $\approx n \times$ PRG, Gen $\approx 2 \times$ Eval
  – Evaluating on the entire domain $[N] \approx N/\lambda \times$ 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 \(\Rightarrow\)
  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⋅λ key size?
  – Amortizing cost of multi-point DPF?
HSS for Branching Programs from DDH
Recall: Homomorphic Secret Sharing

- **Security:** $x^i$ hides $x$
- **Correctness:**
  \[
  \text{Eval}_p(x^1) + \text{Eval}_p(x^2) = P(x)
  \]
δ-HSS

- **Security:** \( x^i \) hides \( x \)
- **δ-Correctness:** Except with prob. \( \delta \) (over Share),
  \[
  \text{Eval}_P(x^1) + \text{Eval}_P(x^2) = P(x)
  \]
Main Theorem

• 2-party $\delta$-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 $\delta$
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.
• $v_i \leftarrow v_j \cdot x_k$ Multiply value in memory by an input.
• Output $v_i \pmod{m}$

We will support homomorphic evaluation of RMS programs over $\mathbb{Z}$ 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:  \(x_1 \times x_2 \times x_3 \times x_4 \ldots x_n\)

Program Output:

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

\[v_i = (1-x_1) v_i + (x_3) v_j + (1-x_1) v_k\]
3 Ways to Share a Number

- Let G be a DDH group of size q with generator g
- 3 levels of encoding \( \mathbb{Z}_q \) elements
  - \([u]\) : \((g^u, g^u) \in G \times G\)
  - \(<v>\) : \((v_1, v_2) \in \mathbb{Z}_q \times \mathbb{Z}_q\) s.t. \(v_1 = v_2 + v\)
  - \({w}\) : \((w_1, w_2) \in G \times G\) s.t. \(w_1 = w_2 \cdot g^w\)
- Each level is additively homomorphic
  - \(<v>,<v'>\) \(\rightarrow\) \(<v+v'>\) \{w\},\{w'\} \(\rightarrow\) \{w+w'\}
- Natural pairing: pair([u],<v>) \(\rightarrow\) \{uv\}
  - \(((g^u)^{v_1}, (g^u)^{v_2}) = (g^{uv_2} \cdot g^{uv}, g^{uv_2})\)
Emulating an RMS program – first attempt:

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

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

$[u] = (g^u, g^u)$

$<v> = (v_2 + v, v_2)$

$\{w\} = (w_2 \cdot g^w, w_2)$

Need Convert: $\{w\} \Rightarrow <w>$

Solved by discrete log...

Stuck?
Share Conversion

Goal: Locally convert multiplicative sharing of $w$ to additive sharing of $w$
Share Conversion

$S$ is a $\delta$-sparse “random” set on $G$

eg $S = \{ h \in G \mid \phi(h) = 0 \}$

for suitable PRF $\phi$

Convert ($g^{zb}$):

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

Goal: Convert multiplicative sharing of $w$ to additive sharing of $w$
Conversion Error

Bad cases:
\[ \exists \bullet \in \text{Bad Zone } \Rightarrow \text{error} \sim \delta_w \]
\[ \not\exists \bullet \in \text{Good Zone } \Rightarrow \text{error} \sim \delta \]

Error probability depends on \( w \)

Las Vegas version
Toy Version

Let’s pretend $g^x$ is a secure encryption of $x$

Emulating an RMS program:

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

\[ [u] = (g^u, g^u) \]
\[ <v> = (v_2 + v, v_2) \]
\[ \{w\} = (w_2 \cdot g^w, w_2) \]
From Toy Version to Real Version

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

$ek = \text{load 1 to memory}$
Applications

• Succinct 2PC for branching programs / logspace / NC¹
  – Communication $|\text{inputs}| + |\text{outputs}| + \text{poly}(\lambda)$ bits

• Sublinear 2PC for “nice” circuits
  – Communication $O(|C|/\log|C|) + \ldots \text{ bits}$
  – $O(|C|)+\ldots$ 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:
  \[ g = 2 \text{ is generator } \& \ p = 2^i - \text{(small)} \]
  \[ h \cdot g = (\text{shift } 1) + \text{ small} \]

• 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