Privacy Preserving Computation in Cloud Using Reusable Garbled Oblivious RAMs

Yongge Wang[1] Qutaibah M. Malluhi[2]

UNC Charlotte, NC 28223, USA

Qatar University, Qatar

Bali, Indonesia, 18-22 December 2022

Outline

A motivating example

- encrypt gmail at Gmail server
- search gmail
  - download gmail to local machine, decrypt it and search
  - search without downloading: search encrypted data (pattern released, some more challenges)
  - how to search over encrypted data without leaking search pattern?

Privacy preserving data storage

- Store encrypted data in cloud
- Carry out computation on these encrypted data: download the encrypted data, process data, and upload
- move computation to the data?
  - computation over encrypted data in the cloud without downloading: does not hide the access pattern to data
  - solution: use oblivious RAM techniques by Goldreich and Ostrovsky for computation over encrypted data, which provably hides all access patterns.

ORAM in cloud

- ORAM schemes require trusted CPUs
- users may not trust the CPU powers at cloud environments
- users run the trusted CPU at client site and to treat the cloud as a large random access memory storage service
- disadvantage: heavy communication overhead between the client and the cloud
- e.g., the most efficient ORAM scheme requires at least $O(\log n)$ memory accesses[1] for each individual memory access, where the cloud database contains n unit blocks of data.

[1] https://web.cs.ucla.edu/ rafail/PUBLIC/128.pdf or https://eprint.iacr.org/2021/1123.pdf

ORAM without trusted CPUs

- Lu and Ostrovsky (2013) and Goldwasser et al (2013): garbled ORAM CPU instead of using trusted CPUs
- disadvantage: the garbled RAM CPU is not succinct and is for one-time use only
- if the ORAM CPU runs t-steps for input x, then the garbled ORAM CPU for the input x is at the size of $O(t)$
- Lu and Ostrovsky asked: Is that possible to use Goldwasser et al's reusable garbled circuits to design reusable garbled ORAMs?
- Goldwasser et al's reusable garbled circuits are based on fully homomorphic encryption (FHE) and Attribute Based Encryption (ABE) schemes, which are not practical

Our approach: Reusable garbled ORAMs

- Intuition: using indistinguishability obfuscation schemes
- our first design (2015): using multilinear map based Jigsaw puzzles:
  - S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: *Proc. IEEE 54th FOCS*
- Unfortuantely, all proposed multilinear maps have been broken
- good news: Jain-Lin-Sahai's (2021) indistinguishability obfuscation schemes
- Our new design: using Jain-Lin-Sahai's technique

A sample ORAM

- the first oblivious RAM by Goldreich (1987): “square root” construction.
- for a RAM machine with n memory cells denoted by an array $R[1..n]$, design an oblivious RAM with a memory array $OR[1..n+2\sqrt{n}]$
- the portion $OR[n+\sqrt{n}+1..n+2\sqrt{n}]$ of size $\sqrt{n}$ is used by the ORAM as the cache space (or a shelter).
- For the first $n+\sqrt{n}$ cells, choose a random permutation

$$\pi : \{1, \cdots, n+\sqrt{n}\} \to \{1, \cdots, n+\sqrt{n}\}$$

and let $OR[\pi(i)] = R[i] = (v_i, x_i)$

A sample ORAM (continued)

- Each time when the ORAM accesses a data block (v_i, x_i) from $OR[\pi(i)] = R[i]$, it stores this value (v_i, x_i) in the cache $OR[n + \sqrt{n} + 1..n + 2\sqrt{n}]$
- For each new query of a data block (v_j, x_j), ORAM checks all values in $OR[n + \sqrt{n} + 1..n + 2\sqrt{n}]$ to see whether (v_j, x_j) has been cached there already. If the data block is found, ORAM only needs to make a dummy access to another cell $OR[\pi(n + l)]$ where l is the counter. That is, this is the l-th dummy memory cell access. If the data block is not found, ORAM loads the data block (v_j, x_j) from $OR[\pi(j)]$ directly.
- After $\sqrt{n}$ memory cell accesses, ORAM needs to re-shuffle data blocks in the memory cells using an oblivious sorting process.

Functional encryption

Definition

A functional encryption scheme $\texttt{FE}$ for a class of functions $\{\mathcal{F}_n\}_{n \in N}$ is a tuple of PPT algorithms

- $(\texttt{fmpk}, \texttt{fmsk}) = \texttt{FE.Setup}(1^\kappa)$ outputs a master public key $\texttt{fmpk}$ and a master secret key $\texttt{fmsk}$
- $\texttt{fsk}_f = \texttt{FE.KeyGen}(\texttt{fmsk}, f)$ outputs a secret key for a function f.
- $c = \texttt{FE.Enc}(\texttt{fmpk}, x)$ outputs a ciphertext for x.
- $y = \texttt{FE.Dec}(\texttt{fsk}_f, c)$ outputs the value y which should equal $f(x)$.

The functional encryption scheme is correct if $y \neq f(x)$ with a negligible probability.

Functional encryption security

Definition

Let $\mathtt{FE}$ be a FE for $\mathcal{F} = \{\mathcal{F}_n\}_{n \in N}$. For a pair of PPT algorithms $A = (A_0, A_1)$ and a PPT simulator S:

$$\begin{array}{l}
\underline{\mathtt{Exp}^{\mathtt{real}}_{\mathtt{FE},A}(1^\kappa):} \\
(\mathtt{fmpk}, \mathtt{fmsk}) \leftarrow \mathtt{FE.Setup}(1^\kappa) \\
(f, \mathtt{state}_A) \leftarrow A_1(\mathtt{fmpk}) \\
\mathtt{fsk}_f \leftarrow \mathtt{FE.KeyGen}(\mathtt{fmsk}, f) \\
(x, \mathtt{state}'_A) \leftarrow A_2(\mathtt{state}_A, \mathtt{fsk}_f) \\
c \leftarrow \mathtt{FE.Enc}(\mathtt{fmk}, x) \\
\mathrm{output}(\mathtt{state}'_\mathtt{A}, \mathtt{c})
\end{array}$$

$$\begin{array}{l}
\underline{\mathtt{Exp}^{\mathtt{ideal}}_{\mathtt{FE},A,S}(1^\kappa):} \\
(\mathtt{fmpk}, \mathtt{fmsk}) \leftarrow \mathtt{FE.Setup}(1^\kappa) \\
(f, \mathtt{state}_A) \leftarrow A_1(\mathtt{fmpk}) \\
\mathtt{fsk}_f \leftarrow \mathtt{FE.KeyGen}(\mathtt{fmsk}, f) \\
(x, \mathtt{state}'_A) \leftarrow A_2(\mathtt{state}_A, \mathtt{fsk}_f \\
\bar{c} \leftarrow S(\mathtt{fmpk}, \mathtt{fsk}_f, f, f(x), 1^{|x|}) \\
\mathrm{output}(\mathtt{state}'_\mathtt{A}, \cancel{\mathtt{c}})
\end{array}$$

FE is secure if there exists a PPT simulator S such that for all pairs of PPT adversaries (A_0, A_1), the outcomes of the two experiments are computationally indistinguishable.

Garbled circuits

Definition

A garbling scheme for a family of circuits $\mathcal{C} = \{\mathcal{C}_n\}_{n\in N}$ is a tuple of PPT algorithms $\texttt{GC} = (\texttt{GC.Garble}, \texttt{GC.Enc}, \texttt{GC.Eval})$ with

- $(\Gamma, \texttt{sk}) = \texttt{GC.Garble}(1^\kappa, C)$ outputs a garbled circuit Γ and a secret key $\texttt{sk}$.
- $c_x = \texttt{GC.Enc}(\texttt{sk}, x)$ outputs an encoding c_x for an input $x \in \{0,1\}^n$.
- $y = \texttt{GC.Eval}(\Gamma, c_x)$ outputs $y = C(x)$.

The garbling scheme $\texttt{GC}$ is *correct* if the probability that $\texttt{GC.Eval}(\Gamma, c_x) \neq C(x)$ is negligible. The garbling scheme $\texttt{GC}$ is *efficient* if the size of Γ is bounded by a polynomial and the run-time of $c = \texttt{GC.Enc}(\texttt{sk}, x)$ is also bounded by a polynomial.

Garbled circuits privacy

Definition

A garbling scheme $\mathtt{GC}$ for a family of circuits $\mathcal{C}$ is said to be *input and circuit private* if there exists a PPT simulator $\mathtt{Sim}_{\mathtt{Garble}}$ such that for all PPT adversaries A and D and all large κ, we have

$$\left| Pr[D(\alpha, x, C, \Upsilon, c) = 1 | \mathrm{REAL}] - Pr[D(\alpha, x, C, \tilde{\Upsilon}, \tilde{c}) = 1 | \mathrm{SIM}] \right| = \mathrm{negl}($$

where REAL and SIM are the following events

$$\begin{aligned} \mathrm{REAL}: & \quad (x, C, \alpha) = A(1^\kappa); (\Upsilon, \mathtt{sk}) = \mathtt{GS.Garble}(1^\kappa, C); c = \mathtt{GS.Enc}(\\ \mathrm{SIM}: & \quad (x, C, \alpha) = A(1^\kappa); (\tilde{\Upsilon}, \tilde{c}) = \mathtt{Sim}_{\mathtt{Garble}}(1^\kappa, C(x), 1^{|C|}, 1^{|x|}). \end{aligned}$$

Reusable garbled circuits

Definition

$\mathtt{RGC}$: a reusable garbling scheme for $\mathcal{C} = \{\mathcal{C}_n\}_{n \in N}$ and $C \in \mathcal{C}_n$.
PPT algorithms $A = (A_0, A_1)$ and PPT simulator $S = (S_0, S_1)$:

$\underline{\mathtt{Exp}^{\mathtt{real}}_{\mathtt{RGC},A}(1^\kappa):}$	$\underline{\mathtt{Exp}^{\mathtt{ideal}}_{\mathtt{RGC},A,S}(1^\kappa):}$
$(C, \mathtt{state}_A) \leftarrow A_0(1^\kappa)$	$(C, \mathtt{state}_A) \leftarrow A_0(1^\kappa)$
$(\mathtt{sk}, \Upsilon) \leftarrow \mathtt{RGC.Garble}(1^\kappa, C)$	$(\tilde{\Upsilon}, \mathtt{state}_S) \leftarrow S_0(1^\kappa, C)$
$\alpha \leftarrow A_1^{\mathtt{RGC.Enc}(\mathtt{sk},\cdot)}(M, \Upsilon, \mathtt{state}_A)$	$\alpha \leftarrow A_1^{O(\cdot,C)[[\mathtt{state}_S]]}(M, \tilde{\Upsilon}, \mathtt{state}$

$\mathtt{RGC}$ is *private* if $\exists S$, $\forall\ A = (A_0, A_1)$, we have

$$\left\{\mathtt{Exp}^{\mathtt{real}}_{\mathtt{RGC},A}(1^\kappa)\right\}_{\kappa \in N} =_c \left\{\mathtt{Exp}^{\mathtt{ideal}}_{\mathtt{RGC},A,S}(1^\kappa)\right\}_{\kappa \in N} \quad (1)$$

Indistinguishability obfuscation

Theorem

(Jain, Lin, and Sahai 2021) Let $\tau > 0$, and $\delta, \varepsilon \in (0, 1)$. Assume the following assumptions with security parameter κ where p is a κ-bit prime, and l, k, n below are large polynomials in κ:

- *LWE assumption over Z_p with subexponential modulus-to-noise ratio 2^{k^ε}, k is dimension of LWE secret*
- *LPN assumption over Z_p with polynomially many LPN samples and error rate $1/l^\delta$, l is dimension of LPN secret*
- *the existence of a Boolean PRG in NC^0 with stretch $n^{1+\tau}$,*
- *SXDH assumption on asymmetric bilinear groups of a order p.*

Then, (subexponentially secure) indistinguishability obfuscation for all polynomial-size circuits exists.

Functional encryption scheme for circuits $C \in NC^1$

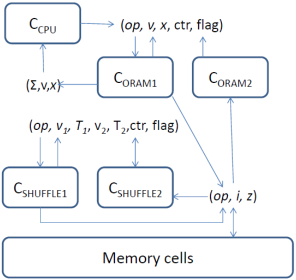
- Choose two standard public-key encryption key pairs $(\mathtt{puk}_1, \mathtt{prk}_1)$ and $(\mathtt{puk}_2, \mathtt{prk}_2)$ in the key generation process of the Functional Encryption scheme.
- The encryption of an input x consists of two ciphertexts of x under the two public keys $\mathtt{puk}_1$ and $\mathtt{puk}_2$ together with a statistically simulation sound non-interactive zero knowledge (NIZK) proof that both ciphertexts encrypt the same message.
- The secret key $\mathtt{sk}_C$ for the circuit C is an indistinguishability obfuscation of a program that first checks the NIZK proof and, if the proof is valid, it uses one of the two secret keys $\mathtt{prk}_1$ and $\mathtt{prk}_2$ to decrypt x and then computes and outputs $C(x)$.

Reusable garbled circuit $\overline{C}$ for a circuit $C \in NC^1$

- Based on Goldwasser et al (2013): "from FE to reusable garbled circuits"
- chooses a secret key $\mathtt{sk}$ to encrypt C as $\mathtt{E.Enc}_{\mathtt{sk}}(C)$.
- $U_E(\mathtt{sk}, x) \in NC^1$ be a UC that decrypts $\mathtt{E.Enc}_{\mathtt{sk}}(C)$ and runs C on x
- $\mathtt{sk}_{U_E}$ be the secret key of the FE scheme for U_E
- The reusable garbled circuit $\overline{C}$ is FE secret key $\mathtt{sk}_{U_E}$.
- The secret key is $(\mathtt{sk}, \mathtt{puk}_1, \mathtt{puk}_2)$
- input to $\overline{C}$ consists of the two cipher texts of $(\mathtt{sk}, x)$ under the two public keys $\mathtt{puk}_1, \mathtt{puk}_2$ and a NIZK proof.
- By combining the results in Theorem 6, De Caro et al (2013), and Goldwasser et al (2013) we have the following result: With assumptions of Theorem 6, there exists a reusable garbling scheme $\mathtt{RGC}$ for circuits in NC^1 that is secure according to the Definition 5 in the random oracle model.

Construction of reusable garbled ORAMs

Figure: Garbled ORAM CPU



$C_{\mathtt{CPU}}$ outputs an encoded $\overline{(op, v, x, \mathtt{ctr}, \mathtt{flag})}$

Inputs: state $(\Sigma, v, x, \mathtt{sctrl}_r)$ and $(\mathtt{session}, \mathtt{sctrl}_s)$.
Output: Encoded $\overline{(\Sigma, v, x)}$, $\overline{\mathtt{session}}$, and interface command $\overline{\mathtt{com}}$

- x,p1 if $\mathtt{sctrl}_r$ and $\mathtt{sctrl}_s$ are inconsistent then exit.
- x,p2 simulate ORAM CPU for one step with state Σ and input (v, x), compute new state Σ and next interface command (op, v, x).
- x,p3 update $\mathtt{session}$, $\mathtt{sctrl}_r$, and $\mathtt{sctrl}_s$.
- x,p4 encode $\mathtt{sk}||(\Sigma, v, x)||\mathtt{sctrl}_r$ to obtain $\overline{(\Sigma, v, x)} = (e_1^{\Sigma}, e_2^{\Sigma}, \pi^{\Sigma})$ as an input to a Goldwasser's reusable garbled circuit
- x,p5 encode $\mathtt{sk}||\mathtt{session}||\mathtt{sctrl}_s$ to obtain $\overline{\mathtt{session}}$ as an input to a Goldwasser's reusable garbled circuit
- x,p6 encode $\mathtt{sk}||(op, v, x, \mathtt{ctr}, \mathtt{flag}, \mathtt{strl}_c)$ to obtain $\overline{\mathtt{com}}$ as an input to a Goldwasser's reusable garbled circuit
- x,p7 output $\overline{(\Sigma, v, x)}$, $\overline{\mathtt{session}}$, and $\overline{\mathtt{com}}$.

C_{ORAM1}

Inputs: $(\Sigma, v, x, \mathtt{sctrl}_r)$, $(op, v, x, \mathtt{ctr}, \mathtt{flag}, \mathtt{sctrl}_c)$, $(\mathtt{session}, \mathtt{sctrl}_s)$

Output: Memory access command (op, i, z)

1. if $\mathtt{sctrl}_c$, $\mathtt{sctrl}_r$, and $\mathtt{sctrl}_s$ are consistent, use $\mathtt{session}$ to update $\mathtt{sctrl}_c$, $\mathtt{sctrl}_r$, and $\mathtt{sctrl}_s$ and go to next step 2. Otherwise, exit
2. if $\mathtt{ctr} < t$, go to step 5.
3. if $\mathtt{ctr} = t$ and $op = \text{READ}$, extract (v, x) from $(op, v, x, \mathtt{ctr}, \mathtt{flag}, \mathtt{sctrl}_c)$ and put it in (Σ, v, x).
4. output $\mathtt{sk}||(\Sigma, v, x, \mathtt{sctrl}_r)$ in the format of input to an Goldwasser's reusable garbled circuit and exit.
5. use $\mathtt{ctr}$, $\mathtt{flag}$, and oblivious memory access to output (op', i, z), where op' and i are in plain text and $z = \overline{(v, x)}$.

C_{ORAM2}

Inputs: (op', i, z), $(op, v, x, \mathtt{ctr}, \mathtt{flag}, \mathtt{sctrl}_c)$, $(\mathtt{session}, \mathtt{sctrl}_s)$
Output: encoded $\overline{\mathtt{sk}||(op, v, x, \mathtt{ctr}, \mathtt{flag}, \mathtt{sctrl}_c, z)}$ using $\mathtt{puk}_1, \mathtt{puk}_2$

- x,p1 if $\mathtt{sctrl}_c$ and $\mathtt{sctrl}_s$ are inconsistent then exit.
- x,p2 use $\mathtt{session}$ to update $\mathtt{sctrl}_c$ and $\mathtt{sctrl}_s$.
- x,p3 decode z to $\mathtt{sk}||(v', x')$ using the key $\mathtt{prk}_1$.
- x,p4 if $v = v'$, then the required data block has been found. Set $\mathtt{flag} =$ yes. Furthermore, if $op =$ READ, insert the value of x' to the x-field of $(op, v, x, \mathtt{ctr}, \mathtt{flag}, \mathtt{sctrl}_c)$.
- x,p5 output encoded $\overline{\mathtt{sk}||(op, v, x, \mathtt{ctr}, \mathtt{flag}, \mathtt{sctrl}_c)}$ in the format of input to an Goldwasser's reusable garbled circuit using public keys $\mathtt{puk}_1, \mathtt{puk}_2$.

Questions

Questions?