PALISADE

Implementing Token-Based Obfuscation under (Ring)LWE

Cheng Chen (MIT -> Gataca)

Nicholas Genise (UCSD->Rutgers->SRI)

Daniele Micciancio (UCSD)

Yuriy Polyakov (Duality and NJIT)

Kurt Rohloff (Duality and NJIT)

WAHC 2020

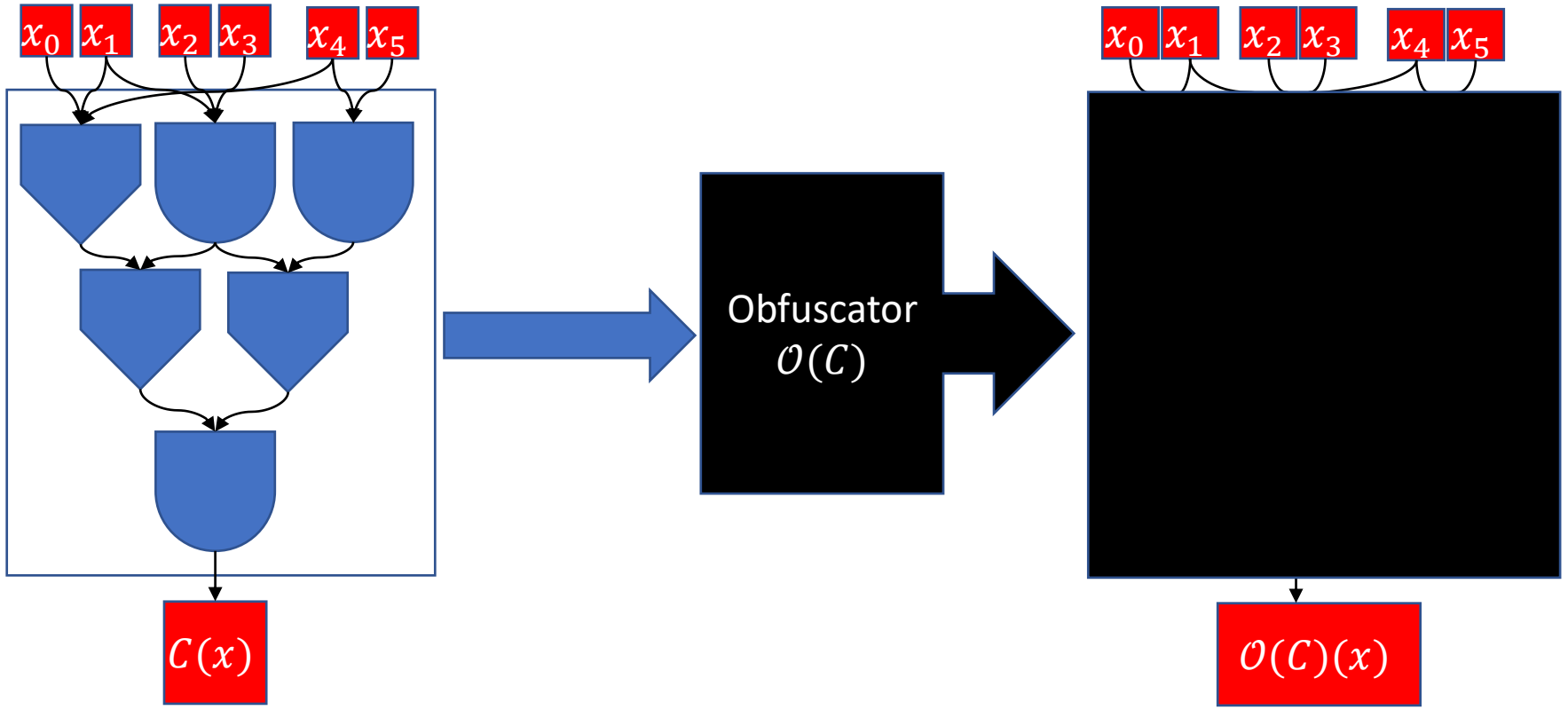
UC San Diego
JACOBS SCHOOL OF ENGINEERING

NJIT
New Jersey Institute of Technology

Motivation

Motivation: Obfuscation

Ideally, we would have a circuit obfuscator: A rigorous way to “encrypt” a circuit while preserving functionality.



Problem: Need to Change the Model

- Classic impossibility result: Virtual Black Box Obfuscation is impossible (assuming OWFs).
- Therefore, we must change the model: allow interaction and limit inputs run by the user.
- Limiting inputs could lead to obfuscating *learnable circuits.*
- Add interaction by adding a vendor who distributes “tokens” on certain inputs.

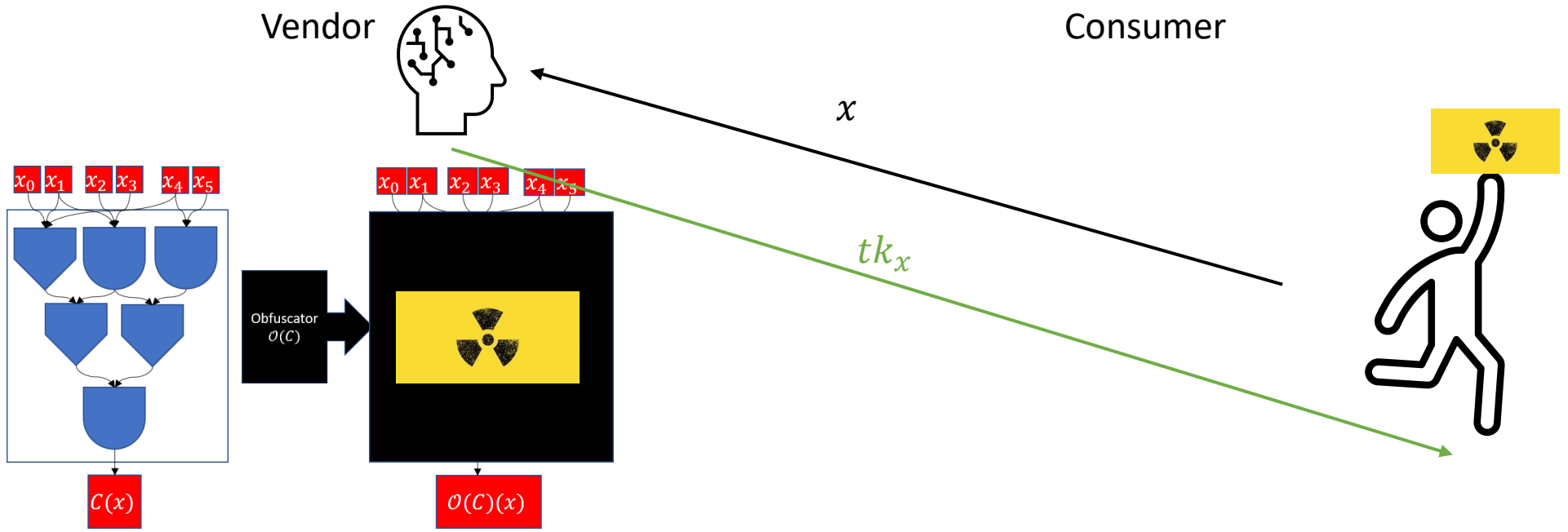
Attempt Using FHE and Yao's Garbled Circuits

- Fully Homomorphic Encryption: Give the user $ct = E_{sk}(x)$ and an evaluation key, ek.
- User runs $E_{sk}\big(C(x)\big) \leftarrow Eval_{ek}(C, ct)$… But can't decrypt!
- Yao's garbled circuits: given an encoding of your input $\mathcal{E}(x)$ and an encoding of the circuit, $\mathcal{E}(C)$, recover $C(x) \leftarrow \mathcal{GC}(\mathcal{E}(x), \mathcal{E}(C))$.
- This is only for a single use!
- Some combination? Garbling FHE decryption fails since the owner must compute $Eval_{ek}(C, ct)$ locally in order to send the correct encoding of $\mathcal{E}(C(x))$ and $\mathcal{E}(C)$.

(QR) Token-Based Obfuscation (TBO)

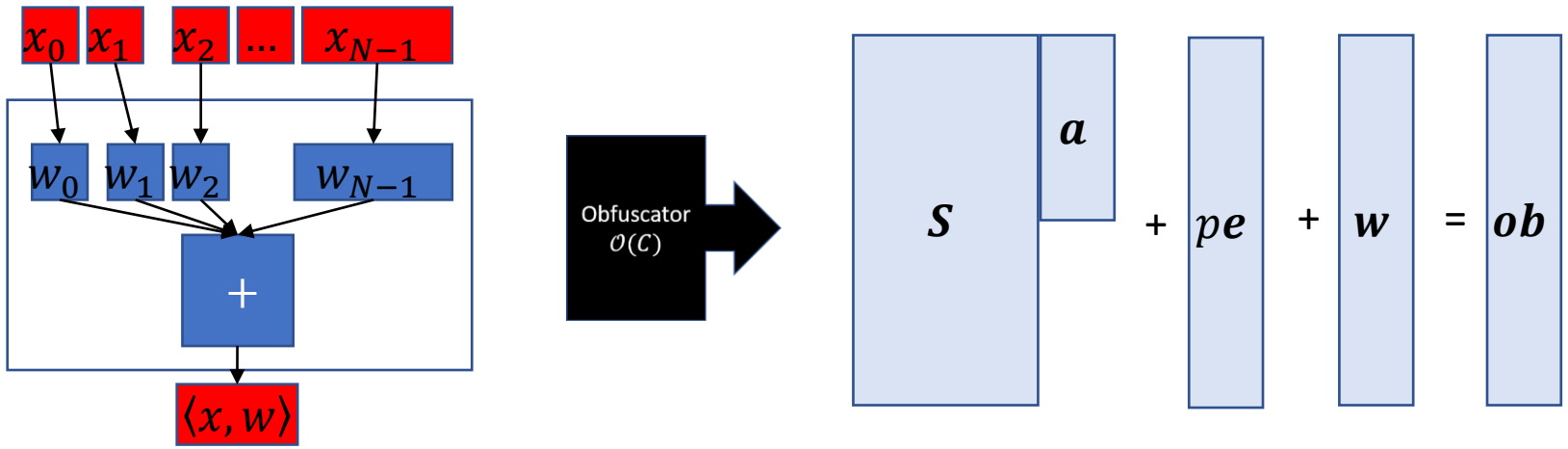
TBO was introduced by Goldwasser et. al. (STOC 2013) as an achievable form of obfuscation obtainable from functional encryption and circuit garbling.

Tokens can be generated before circuit obfuscation (precomputed).



Linear Functions

Warm-Up: Linear Functions from LWE



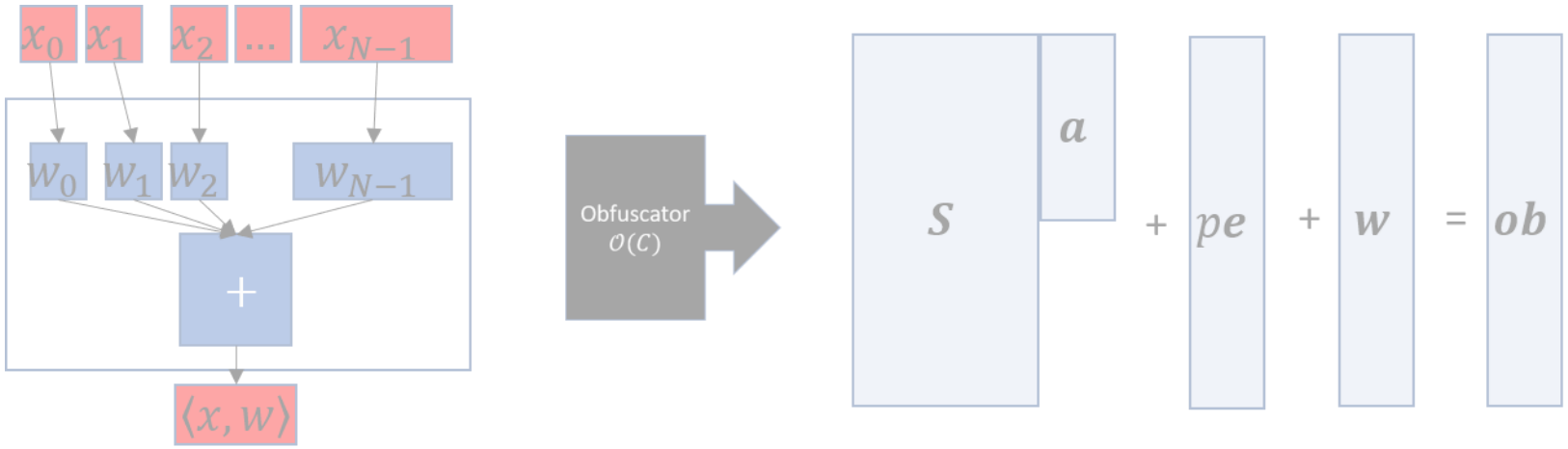
$C(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{w} \rangle \; mod \; p.$

Encode the weights (circuit) with LWE with a modulus larger than p.

Tokens are $\boldsymbol{t}_x = \boldsymbol{x}^t \boldsymbol{S}$ and compute with $\langle \boldsymbol{x}, \boldsymbol{ob} \rangle - \langle \boldsymbol{a}, \boldsymbol{t}_x \rangle \; mod \; p.$

Properties: limit number of tokens, but tokens are oblivious to the circuit.

Linear Functions from LWE: Implementation



$N = 4k, n = 2048, log_2(q) = 52, log_2(p) = 25,$ binary $\mathbf{w}$

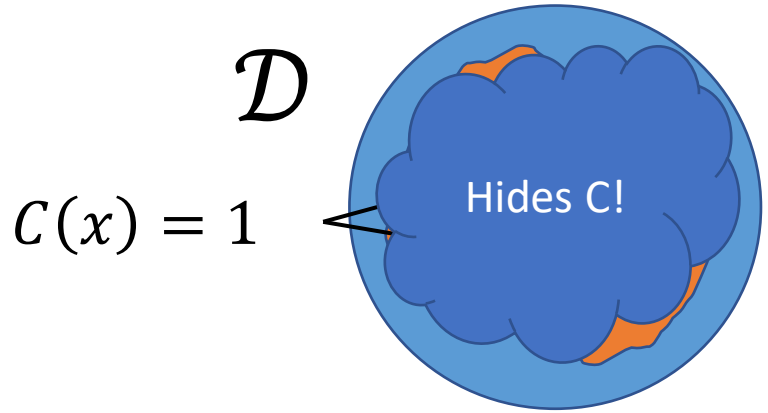
Run on an i7 Processor, 4 cores @ 3.4 GHz, 16 GB of RAM

Mode	Key [KB]	Program size [KB]	KeyGen [ms]	Obf. [ms]	TokenGen [ms]	Eval [ms]
w/ AES	0.023	31.3	0.40	515	492	0.27
w/o AES	64,032	31.3	1,204	78.3	55.9	0.27

Conjunctions and Branching Programs

CHCPRFs: Definition

- In general, our constructions follow from constraint-hiding constrained PRFs (CHCPRFs).
- CHCPRFs are PRFs $\{\mathcal{F}_K | \mathcal{F}_K \colon \mathcal{D} \to \mathcal{R}\}$ with a master K_{msk} which constrained keys can be derived from: K_C.
- This constrained key corresponds to a circuit (predicate) on the PRF's input space: $C \colon \mathcal{D} \to \{0,1\}$.
- Then, $\mathcal{F}_{K_C}(x) = \mathcal{F}_{K_{msk}}(x)$ when $C(x) = 1$ and $\mathcal{F}_{K_C}(x) \approx_c \mathcal{U}(\mathcal{R})$.



QR-TBO from CHCPRFs

- The vendor samples a PRF: $\mathcal{F}_{msk}$
- To obfuscate a circuit, the vendor generates a constrained key: $\mathcal{F}_{K_C}$.
- A token for x is simply the function evaluated at x: $tk_x \leftarrow \mathcal{F}_{msk}(x)$.
- The user evaluates C by checking $tk_x = \mathcal{F}_{K_C}(x)$?

- Why is this QR-TBO? We reveal the input to the vendor while the circuit C stays hidden!

Conjunction Obfuscation (Linear Classifier) Challenge

- Image recognition
  - Goal: determine if an image is “close enough” to a given “pattern.”

- Feature Extraction approach
  - Identify and extract binary features from images
  - Compare closeness of features to a reference pattern to determine a match.

- (Informal) Security goal:
  - Want to deploy this classifier but protect the pattern from being extracted by adversaries.

Conjunction Implementation

- We optimized and implemented Canetti and Chen's 2017 CHCPRF for conjunctions (bit fixing represented by $\boldsymbol{c} \in \{0,1,*\}$) in PALISADE.
- CC17's construction uses GGH15 + (R)LWR in a provable manner.

- Contributions:
  - Larger, non-binary, alphabet for encoding (8-bit word size). This reduces the GGH-15 depth.
  - Adapted GM'18 and GMP'19's DG sampling to larger GLWE dimension.
  - Security under RLWE (compared to *entropic* RLWE)
  - Tighter correctness analysis -> smaller functional parameters (e.g. subgaussian error analysis)
  - More efficient TokenGen and Eval due to CRT-based scaling and rounding.

Conjunction Implementation, Comparison

- Run on a server with 2 Intel(R) Xeon(R) E5 2680 v4 @ 2.4 GHz, 14 cores each and a total of 500 GB of RAM.

Table 2: Execution times and program size for conjunction obfuscation; $\lambda \geq 80$.

L [bits]	# threads	n	$\lceil \log_2 q \rceil$	$\log_2 t$	Program size [GB]	Obf. [min]	TokenGen [ms]	Eval [ms]	EvalTotal [ms]
Token-Based Obfuscation									
32	1	4096	180	20	11.6	23.5	1.3	75.8	77.1
32	14	4096	180	20	11.6	5.1	0.6	11.0	11.6
64	28	8192	360	20	300	52.5	4.0	269.9	273.9
Optimized Distributional VBB Obfuscation [27]									
32	14	4096	180	15	36.8	12.4	–	–	53.0

Branching Program Implementation

- We optimized and implemented CVW 2019 CHCPRF for Branching Programs in PALISADE.
- Implemented both general BPs and Permutation BPs. (General BP >> Perm)
- Application: Hamming weight threshold function.

- Contributions:
  - Larger, non-binary, alphabet for encoding (8-bit word size). This reduces the GGH-15 depth.
  - Adapted GM'18 and GMP'19's DG sampling to larger GLWE dimension.
  - Tighter correctness analysis -> smaller functional parameters (e.g. subgaussian error analysis)
  - More efficient TokenGen and Eval due to CRT-based scaling and rounding.

BP Implementation, Comparison

- Run on a server with 2 Intel(R) Xeon(R) E5 2680 v4 @ 2.4 GHz, 14 cores each and a total of 500 GB of RAM.

Table 3: Execution times and program size for TBO of the branching program checking whether two 24-bit strings (one of them is obfuscated) have a Hamming distance less than T; # threads $= 28$, $n = 4096$, $\lceil \log_2 q \rceil = 180$, $\log_2 t = 20$, $\lambda \geq 80$.

T	d	Program size [GB]	Obf. [min]	TokenGen [ms]	Eval [ms]
1	3	76.6	26.9	0.6	55.0
2	4	136	44.8	0.6	66.6
3	5	213	72.6	0.9	133

Thank you!

- Questions?
- Our full paper is at https://eprint.iacr.org/2018/1222.
- https://gitlab.com/palisade/palisade-development