Token-Based Obfuscation of Conjunction Programs using Constraint-Hiding Constrained Pseudorandom Functions

Cheng Chen, Yuriy Polyakov, Kurt Rohloff, and Vinod Vaikuntanathan

February 21, 2018

1 Conceptual Model

Token-based obfuscation (TBO) of conjunction programs based on constrained-hiding constrained pseudorandom functions (CHCPRF) uses the same building blocks as the distributed virtual black-box (DVBB) obfuscation of conjunction programs implemented in [1]. The construction we implemented is an optimized variant of the bit-fixing CHCPRF construction described in section 5.1 of [2]. The construction of [2] is secure under Ring LWE (in contrast to entropic Ring LWE for the DVBB construction in [1]). Our construction is also secure under Ring LWE (using the small-secret ring elements generated by uniformly random distribution over $\{-1, 0, 1\}^n$).

The full workflow (when applied to the TBO scenario) includes the following stages:

- Parameter generation (**ParamGen**): generate lattice parameters based on the length of the pattern and security level.
- Key generation (**KeyGen**): Generate trapdoor key pairs and an unconstrained master secret key. The unconstrained key corresponds to a pattern of all wild cards (accepts any pattern).
- Obfuscation of a given pattern (**Constrain**): An obfuscated program (constrained key) is generated by replacing the master key elements with random samples where a specific bit is fixed (no changes are made for wild cards in the input pattern).
- Evaluate using the constrained key (**EvalD**): Evaluates the obfuscated program and outputs a vector $\mathbf{y} \in R_p^{1\times m}$, where $\mathcal{R}_p = \mathbb{Z}_p[x]/\langle x^n + 1\rangle$.
- Evaluate using the master (unconstrained) key (**EvalS**): Compute a vector $\mathbf{y}' \in R_p^{1\times m}$.
- Comparison test (**EqualTest**): Compare $\mathbf{y}$ with $\mathbf{y}'$; if they match, output 1 (True), otherwise output 0 (False).

The output of **EvalS** is the PRF value, and is used as the "token" in our case. If the token for the unconstrained key (master seret key) matches the output for the constrained key (obfuscated program), the result is 1 (True).

The token-based model involves a trusted party (hardware) that stores the MSK and runs **EvalS**. The **EvalD** operation is executed by a public (untrusted) party.

ParamGen, **KeyGen**, and **Constrain** are offline operations.

EvalD, **EvalS**, and **EqualTest** are online operations. The evaluation runtime is the sum of the runtimes for the online operations.

The main difference of the TBO model as compared to the DVBB model [1] is the interaction between an untrusted and trusted components of the system. This bounds the number of evaluation queries and prevents exhaustive search attacks that the DVBB construction is amenable to.

2 ALGORITHMS FOR TBO FUNCTIONS

We use the same notation as in [1]. All algorithmic optimizations from [1] have also been applied.

The key generation algorithm is listed in Algorithm 1. Parameter $\mathcal{L}$ is the effective length of conjunction pattern. As compared to the construction in [2], we optimized the master secret key generation to only sample short ring elements $s_{i,b}$ (without calling the trapdoor sampling for these short ring elements).

Algorithm 1 Key generation

function KEYGEN(1^λ)
  for i = 0..$\mathcal{L}$ **do**
    $\mathbf{A}_i, \widetilde{\mathbf{T}}_i := \text{TRAPGEN}(1^\lambda)$
  end for
  for i = 1..$\mathcal{L}$ **do**
    for b = 0..2^w-1 **do**
      $s_{i,b} \leftarrow \mathcal{T}$
    end for
  end for
  return $K_{MSK} := \left(\{s_{i,b}\}_{i\in\{1,..,\mathcal{L}\},b\in\{0,..,2^w-1\}}, \{\mathbf{A}_i, \widetilde{\mathbf{T}}_i\}_{i\in\{0,..,\mathcal{L}\}}\right)$
end function

The conjunction obfuscator relies on the **Encode** algorithm of directed-encoding ring instantiation to encode each word of the conjunction pattern. The **Encode** algorithm is depicted in Algorithm 2.

Algorithm 2 Directed encoding

function $\mathsf{Encode}_{\mathbf{A}_i\to\mathbf{A}_{i+1}}(\mathbf{T}_i, r, \sigma)$
  $\mathbf{e}_{i+1} \leftarrow \mathcal{D}_{\mathcal{R},\sigma} \in \mathcal{R}_q^{1\times m}$.
  $\mathbf{b}_{i+1} := r\mathbf{A}_{i+1} + \mathbf{e}_{i+1} \in \mathcal{R}_q^{1\times m}$
  $\mathbf{R}_{i+1} := \mathsf{GaussSamp}(\mathbf{A}_i, \mathbf{T}_i, \mathbf{b}_{i+1}, \sigma_t, s) \in \mathcal{R}_q^{m\times m}$
  return $\mathbf{R}_{i+1}$
end function

Algorithm 3 lists the pseudocode for the main obfuscation function. Just like in [1], we encode words of conjunction pattern $\mathbf{v} \in \{0, 1, \star\}^L$. Each word is w bits long, and 2^w is the number of encoding matrices for each encoded word of the pattern. The actual pattern length L gets replaced with the effective length $\mathcal{L} = \lceil L/w \rceil$ to reduce the number of encoding levels (multlinearity degree).

The $s_{i,b}, r_{i,b}$ elements are ternary uniformly random ring elements, i.e., sampled over $\{-1, 0, 1\}^n$, for $i \in [\mathcal{L}]$ and $b \in \{0, \ldots, 2^w - 1\}$.

Algorithm 4 shows the psedocode for the evaluation using the obfuscated program (constrained key).

Algorithm 5 lists the pseudocode for **EvalS**, the evaluation using unconstrained key. Our variant is significantly optimized compared to the construction in [2]: it multiplies short ring

elements followed by a single scalar product with the public key $\mathbf{A}_0$ (in contrast to vector-matrix products in [2]).

Algorithm 3 Obfuscation

function CONSTRAIN($\mathbf{v} \in \{0,1,*\}^L, K_{MSK}, \sigma$)
  for `i = 1..`$\mathcal{L}$ **do**
    `Build binary mask` M (0's correspond to wild-card bits, 1's correspond to fixed bits)
    for b `= 0..`2^w`-1` **do**
      if $(b \wedge M) \neq (v \wedge M)$ **then**
        $r_{i,b} \leftarrow \mathcal{T}$
      else
        $r_{i,b} := s_{i,b}$
      end if
      $\mathbf{D}_{i,b} := \mathsf{Encode}_{\mathbf{A}_{i-1} \to \mathbf{A}_i}(\widetilde{\mathbf{T}}_{i-1}, r_{i,b}, \sigma)$
    end for
  end for
  $\Pi_v := \left(\mathbf{A}_0, \{\mathbf{D}_{i,b}\}_{i \in [\mathcal{L}], b \in \{0,\ldots,2^w-1\}}\right)$
  return Π_v
end function

Algorithm 4 Evaluation using the obfuscated program

function EVALD($\mathbf{x} \in \{0,1\}^L$, Π_v)
  $\mathbf{D}_{\mathbf{\Pi}} := \mathbf{A}_0$
  for `i = 1..`$\mathcal{L}$ **do**
    $\mathbf{D}_{\mathbf{\Pi}} := \mathbf{D}_{\mathbf{\Pi}}\, \mathbf{D}_{i,x[1+(i-1)w\,:\,iw]} \in {\mathcal{R}_q}^{1 \times m}$
  end for
  $\mathbf{y} := \lfloor \frac{2}{q} \mathbf{D}_{\mathbf{\Pi}} \rceil$
  return $\mathbf{y}$
end function

Algorithm 5 Evaluation by a trusted party (using the master secret key)

function EVALS($\mathbf{x} \in \{0,1\}^L$, K_{MSK})
  $\Delta := \mathbf{A}_0 \prod_{i=1}^{\mathcal{L}} s_{i,x[1+(i-1)w\,:\,iw]}$
  $\mathbf{y}' := \lfloor \frac{2}{q} \Delta \rceil$
  return $\mathbf{y}'$
end function

3 PARAMETER SELECTION

The correctness constraint for a conjunction pattern with $\mathcal{L}$ words ($\mathcal{L} \geq 2$) is expressed as

$$q > 16 B_e \left(\beta s \sqrt{mn}\right)^{\mathcal{L}-1}, \tag{1}$$

where $B_e = 6\sigma, \beta = 6$, and all other parameters are the same as in [1].

The correctness constraint (1), which was derived using the Central Limit Theorem, significantly reduces bitwidth requirements for modulus q (as compared to the analysis in [2]). The derivation details are very similar to the DVBB case (see Appendix B of [1]).

REFERENCES

[1] D. B. Cousins, G. D. Crescenzo, K. D. Gür, K. King, Y. Polyakov, K. Rohloff, G. W. Ryan, and E. Savaş, "Implementing conjunction obfuscation under entropic ring lwe," in *2018 IEEE Symposium on Security and Privacy (SP)*, vol. 00, pp. 68–85, cryptology ePrint Archive, Report 2017/844, https://eprint.iacr.org/2017/844. [Online]. Available: doi.ieeecomputersociety.org/10.1109/SP.2018.00007

[2] R. Canetti and Y. Chen, "Constraint-hiding constrained prfs for nc1 from lwe," Cryptology ePrint Archive, Report 2017/143, 2017, https://eprint.iacr.org/2017/143.