Sparse Graph Obfuscation

September 26, 2023

1 Preliminaries

1.1 Bounded Functional Encryption

We will use the notation of static, bounded functional encryption as presented in [AR17].

Security

We will slightly weaken the security notion such that the adversary does not choose which circuits it can learn the functional secret key for. Indeed, this is a weaker notion of functional encryption which fixes the adversary's output circuit. We will assume that we get circuit $C_1, \ldots, C_d$.

See page 8 of [AR17] for now. I'll put in the actual definition later.

$\mathrm{Exp}^{\mathrm{real}}_{\mathcal{F},\mathcal{A}}(1^\lambda)$

$(\mathrm{MPK}, \mathrm{MSK}) \leftarrow \mathrm{FE.Setup}(1^\lambda)$
$\mathrm{SK}_{C_i} \leftarrow \mathrm{FE.Keygen}(\mathrm{MSK}, C_i)$ for $i \in [d]$
$x_i \leftarrow \mathcal{A}(\mathrm{SK}_{C_i})$
$\mathrm{CT}_i \leftarrow \mathrm{FE.Enc}(\mathrm{MPK}, x_i)$
$\alpha \leftarrow \mathcal{A}(\mathrm{CT}_1, \ldots, \mathrm{CT}_d)$
return $x_1, \ldots, x_d, \alpha$

$\mathrm{Exp}^{\mathrm{ideal}}_{\mathcal{F},\mathrm{Sim}}(1^\lambda)$

$(\mathrm{MPK}, \mathrm{MSK}) \leftarrow \mathrm{FE.Setup}(1^\lambda)$
$\mathrm{SK}_{C_i} \leftarrow \mathrm{FE.Keygen}(\mathrm{MSK}, C_i)$ for $i \in [d]$
$x_i \leftarrow \mathcal{A}(\mathrm{SK}_{C_i})$
$\mathrm{CT}_i \leftarrow \mathrm{Sim}(1^\lambda, 1^{|x_i|}, \mathrm{MPK}, C_i, \mathrm{SK}_{C_i}, C_i(x_i))$
$\alpha \leftarrow \mathcal{A}(\mathrm{CT}_1, \ldots, \mathrm{CT}_d)$
return $x_1, \ldots, x_d, \alpha$

Note that the adversary $\mathcal{A}$ and simulator are stateful but we do not include this in the above notation for simplicity.

2 A sketch for the boys

2.1 Graph Label Randomization

Say that we have a sparse graph $\mathcal{G} = (V, E)$ such that $|V| = n$ and $\forall v \in V, \deg(v) = d$. (TODO: padding).

Then, we want to create a pseudo-randomized label mapping of the graph, $\phi : \{0,1\}^\lambda \times V \to \{0,1\}^{c \cdot \lambda}$ such that ϕ is deterministic and pseudo-random. In particular, we require that for an adversary that does not know a path from v to $u \in \{v_1, ..., v_p\}$ or u to v where $v \neq u$, then for $K, K' \in \{0,1\}^\lambda$,

$$\begin{aligned} &\mathbf{Pr}\left[\mathcal{A}(\phi(K, v), \phi(K, v_1), \dots \phi(K, v_p), v_1, \dots, v_p, C_\Gamma) = 1\right] \\ &- \mathbf{Pr}\left[\mathcal{A}(\phi(K', v), \phi(K, v_1), \dots \phi(K, v_p), v_1, \dots, v_p, C_\Gamma) = 1\right] \leq \mathtt{negl}(\lambda) \end{aligned} \tag{1}$$

where C_Γ is the neighbor function for the embedded space: i.e. $C_\Gamma = \phi \circ \Gamma \circ \phi^{-1}$.

2.2 The Construction

Algorithm 1 The circuit for the neighbor function, C_Γ.

function $\mathtt{INNER}_i(\mathtt{Dec}(\phi(v)) = v, K)$
  $u_1, \dots, u_d = \Gamma(v)$
  $u = u_i$
  $r = \mathtt{PRF}(K, u)$
  return FE.Enc(MPK, (u, K)) where we encrypt with randomness from r.
function $C_\Gamma(\phi(v))$
  for $i \in [d]$ **do**
    $u_i = \text{FE.Dec}(\text{SK}_{\mathtt{inner}_i}, \phi(v))$
  return $(u_1, \dots, u_d)$

Claim 2.1. *eq. (1) holds for a given vertex v for any PPT adversary, $\mathcal{B}$ when C_Γ is implemented as in algorithm 1.*

In order to prove the above, we must show that an adversary which does not know a path from v to $v_1, \dots v_p$ essentially "learns" nothing of K.

Lemma 2.2 (Does not learn K). *For any PPT adversary $\mathcal{A}$*

$$\begin{aligned} \Big| &\mathbf{Pr}[\mathcal{A}(\phi(K, v_1), \dots, \phi(K, v_p), v_1, \dots, v_p, C_\Gamma) = 0] \\ &- \mathbf{Pr}[\mathcal{A}(\phi(K', v_1), \dots, \phi(K', v_p), v_1, \dots, v_p, C_\Gamma) = 0]\Big| \leq \mathtt{negl}(\lambda). \end{aligned} \tag{2}$$

Proof. We proceed via showing that for any calls to C_Γ, the output is computationally indistinguishable from an output where the adversary is given $\phi(K', v_1), \dots, \phi(K', v_p)$ for $K' \in \{0,1\}^\lambda$.

We proceed via a hybrid argument.

- $\mathtt{Hyb}_0$: The adversary plays the game outlined in section 1.1 where the circuits are $\mathtt{inner}_1, \dots, \mathtt{inner}_d$
- $\mathtt{Hyb}_2$: As the above except that we replace the real protocol with the simulated one, section 1.1

- $\texttt{Hyb}_3$: As the above except that we replace $\texttt{inner}_i$ with $\texttt{inner}'_i$ where $\texttt{inner}'_i$ invokes the FE simulator, Sim when generating its output. Note that Sim must know the access pattern of $\mathcal{A}$ to $\texttt{inner}_i$ in order to simulate the output of $\texttt{inner}_i$. So, we replace $\texttt{inner}'_i$ working backwards from the last call to the simulator to the first. As the simulator is stateful, we can see that the simulator knows the access pattern.

- $\texttt{Hyb}_4$: As the above except that we replace K with K'. We can see that this is valid as both inputs to $\texttt{inner}'_i$ and outputs of $\texttt{inner}'_i$ are independent of K.

From the above hybrid, we can see that if $\mathcal{A}$ can distinguish eq. (2), then $\mathcal{A}$ can distinguish $\texttt{Hyb}_4$ and $\texttt{Hyb}_3$. □

Now we can prove Claim 2.1,

Proof of Claim 2.1. First note that the adversary cannot learn $\phi(K, v)$ via calling FE.Enc(K, v) as that would imply breaking lemma 2.2.

Thus, $\mathcal{B}$ can only learn $\phi(K, v)$ via calling C_Γ or manipulating given cipher texts. We now proceed to show that this is computationally infeasible via a hybrid algorithm.

- $\texttt{Hyb}_0$: The adversary plays the game outlined in section 1.1 where the circuits are $\texttt{inner}_1, \ldots, \texttt{inner}_d$ and $\mathcal{A}$ is given $\phi(v_1) = \text{FE.Enc}(v_1, K), \ldots, \phi(v_p) = \text{FE.Enc}(v_p, K)$ where encryption randomness is derived from $\texttt{PRF}(K, v_1), \ldots, \texttt{PRF}(K, v_p)$.

- $\texttt{Hyb}_1$: As the above except that we replace encryption randomness with true random strings fixed for each v_ℓ where $\ell \in [p]$.

- $\texttt{Hyb}_2$: As the above except that we replace $\texttt{inner}_i$ with $\texttt{inner}'_i$ such that if $\texttt{inner}_i(\phi(K, u)) = \phi(K, v)$, $\texttt{inner}'_i(\phi(K, u)) = \perp$. Note that $\mathcal{A}$ cannot distinguish between $\texttt{inner}_i$ and $\texttt{inner}'_i$ because $\mathcal{A}$ does not know the path from v_ℓ to v and can thus not find $\phi(K, u)$ from repeated queries of $\texttt{inner}_i$.

- $\texttt{Hyb}_3$: As the above except that we replace section 1.1 with section 1.1, the simulated version. Now, note that $\mathcal{A}$ cannot distinguish between simulated $\phi(K, v_\ell)$ which we will call $\phi(v_\ell)'$ and the given $\phi(v_\ell)$.

Indeed the simulated labels, $\phi'(K, v_\ell)$ are simulated independently of $\phi(K, v)$ as $\texttt{inner}'_i(K, u) \neq \phi(K, v)$ with high probability for any $u \in V$. Thus,

$$\left| \mathbf{Pr}[\mathcal{B}(v, C_\Gamma) = \phi(K, v)] - \mathbf{Pr}[\mathcal{B}\big(v, \phi(K, v_1), \ldots, \phi(K, v_p), v_1, \ldots, v_p, C_\Gamma\big) = \phi(K, v)] \right| \leq \texttt{negl}(\lambda)$$

And, by lemma 2.2, we have that

$$\mathbf{Pr}[\mathcal{B}(v, C_\Gamma) = \phi(K, v)] \leq \texttt{negl}(\lambda)$$

thus concluding the proof. □

September 26, 2023

Abstract

References

[AR17] Shweta Agrawal and Alon Rosen. Functional encryption for bounded collusions, revisited. In *Theory of Cryptography Conference*, pages 173–205. Springer, 2017. 1.1, 1.1