Streaming Functional Encryption

Jiaxin Guan[*] Alexis Korb[†] Amit Sahai[‡]

November 2022

Abstract

We initiate the study of *streaming functional encryption* (sFE) which is designed for scenarios in which data arrives in a streaming manner and is computed on in an iterative manner as the stream arrives. Unlike in a standard functional encryption (FE) scheme, in an sFE scheme, we (1) do not require the entire data set to be known at encryption time and (2) allow for partial decryption given only a prefix of the input. More specifically, in an sFE scheme, we can sequentially encrypt each data point x_i in a stream of data $x = x_1 \dots x_n$ as it arrives, without needing to wait for all n values. We can then generate function keys for streaming functions which are stateful functions that take as input a message x_i and a state st_i and output a value y_i and the next state st_{i+1}. For any $k \leq n$, a user with a function key for a streaming function f can learn the first k output values $y_1 \dots y_k$ where $(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$ and $\mathsf{st}_1 = \perp$ given only ciphertexts for the first k elements $x_1 \dots x_k$.

In this work, we introduce the notion of sFE and show how to construct it from FE. In particular, we show how to achieve a secure sFE scheme for $\mathsf{P/Poly}$ from a compact, secure FE scheme for $\mathsf{P/Poly}$, where our security notion for sFE is similar to standard FE security except that we require all function queries to be made before the challenge ciphertext query. Furthermore, by combining our result with the FE construction of Jain, Lin, and Sahai (STOC, 2022), we show how to achieve a secure sFE scheme for $\mathsf{P/Poly}$ from the polynomial hardness of well-studied assumptions.

[*]Princeton University. Email: `jiaxin@guan.io`.
[†]UCLA. Email: `alexiskorb@cs.ucla.edu`.
[‡]UCLA. Email: `sahai@cs.ucla.edu`.

Contents

1 Introduction **1**
1.1 Our Results . . . 2
1.2 Related Work . . . 5

2 Technical Overview **6**
2.1 Single-Key, Single-Ciphertext, SIM-Secure, Secret-Key Streaming FE . . . 6
2.2 Bootstrapping to an IND-Secure, Public-Key Streaming FE . . . 13

3 Preliminaries **16**
3.1 Functional Encryption . . . 16

4 Streaming Functional Encryption **20**
4.1 Security . . . 21

5 Single-Key, Single-Ciphertext, SIM-secure, Secret-Key Streaming FE **24**
5.1 Parameters . . . 25
5.2 Construction . . . 28
5.3 Correctness and Efficiency . . . 30
5.4 Security . . . 32

6 Bootstrapping to an IND-Secure, Public-Key Streaming FE Scheme **54**
6.1 Parameters . . . 55
6.2 Construction . . . 57
6.3 Correctness and Efficiency . . . 59
6.4 Security . . . 60

7 Acknowledgements **60**

8 References **61**

A [JLS22] Assumptions **64**

B Preliminaries Continued **65**
B.1 Standard Notions . . . 65
B.2 Secret-Key Functional Encryption . . . 67

C Additional Streaming FE Definitions **69**
C.1 Secret-Key Streaming FE . . . 69
C.2 Relaxed Definition of Streaming FE . . . 70

D Security Proof from Section 6 **71**

1 Introduction

Functional encryption (FE) [SW05, BSW11, O'N10] is a powerful extension of public key encryption that restricts users with secret keys to only learning functions of the encrypted data. In an FE scheme, an authority can generate function keys for functions of their choice using a master secret key. Given a function key for f and an encryption of x, one should be able to learn $f(x)$ and nothing else. Functional encryption has been studied extensively (e.g. [SW05, GGH$^+$13, SW14, GGHZ16, GKP$^+$13, BGG$^+$14, GVW15, ABSV15, AJ15, BV15, Lin16, Lin17, GPSZ17, GPS16, LV16, AS17, LT17, AJS18, AJL$^+$19, Agr19, JLMS19].) In addition to its many direct applications, FE has also been used to build other cryptographic applications such as reusable garbled circuits [GKP$^+$13], adaptive garbling [HJO$^+$16], multi-party non-interactive key exchange [GPSZ17], universal samplers [GPSZ17], and verifiable random functions [GHKW17, Bit17, BGJS17]. Importantly, FE can be used to construct $i\mathcal{O}$ [BV15, AJ15], a powerful tool which can be used to build many cryptographic primitives [SW14].

Now is an exciting time for functional encryption. While early constructions of FE were restricted – for example, some required a bound on the number of function keys [GVW12], or only allowed functions keys for simple functions like inner product [ABDP15, ALS16] or quadratic functions [BCFG17] – we've recently been able to achieve FE for $\mathsf{P/Poly}$ from well-studied assumptions [JLS21, JLS22]. This has also opened the door to extensions such as FE for Turing machines [AS16] and multi-input FE [GGG$^+$14]. In light of these advances, it is natural to consider the feasibility of even stronger notions of functional encryption.

The Streaming Scenario. In many modern applications, the data sets being used might not be available all at once or might be in some ongoing process of being generated. Additionally, data sets are often large, and it can be difficult to store or compute on the entire data set all at once. Using functional encryption in these scenarios can incur a large expense or may not even be possible.

For example, consider a privacy-preserving machine learning algorithm that is being trained on a massive data set provided by a third party. The third party might hope to use FE to protect the training data by encrypting it and providing it to the training algorithm user along with a function key for the algorithm. However, using FE in this manner requires the training set to be fixed at encryption time. If new training data later becomes available, the user cannot continue training the algorithm on this data without re-encrypting the entire data set. Furthermore, the user cannot generate any partial results while training the algorithm but must instead wait until the full decryption finishes, which takes time and space proportional to the size of the data set.

Using FE in these scenarios is additionally infeasible when the data arrives in a streaming fashion either due to the nature of the data or because the data is too large to be stored on the user's computer all at once. As an example, consider a video-processing algorithm. For privacy, the video broadcaster might consider using FE to send an encryption of the video and a function key for the video-processing algorithm to the user. However, if the video is being recorded live or is large in size, then we would ideally like the broadcaster to be able to stream an encryption of the video to the user who could then begin processing the video as the stream arrives. However, this is not possible with regular FE. The broadcaster would have to wait until the video is finished (if it ever is!) to encrypt the video, and then send the entire encryption to the user, who could only then begin processing the video. Furthermore, the user would have to compute on an encryption of the entire video stream, which may be large.

As another example, consider a business that receives data from many internet users. Suppose that an outside company wishes to run an algorithm on this data. To protect the data of the internet users, the business could use FE to send a function key of the algorithm to the other company along with an encryption of the internet users' data. As the internet users are not likely to be concurrently online, the data is unlikely to be available all at once. Ideally, the business could collect, encrypt, and send the data as it becomes available to them, without needing to store it long term. However, if we are using regular FE, then the business would have to store all of the received data until a time when it has received sufficient data from a sufficient number of internet users. Only then could the business encrypt the data and send it to the outside company. At this point, the data set provided to the outside company is fixed, and adding new data to the set is difficult and may require re-encrypting all of the data. As this data set may be very large, it may also be difficult for the business or the outside company to store the data in its entirety or compute FE functionalities on it.

The reason that FE is so expensive in these scenarios is that when using FE, the entire data set must be known at encryption time, and decryption can only be run on a ciphertext for the entire data set. To counter these issues, we put forward a new type of FE which is better suited for these scenarios.

1.1 Our Results

In this work, we introduce the notion of streaming functional encryption (sFE) and show how to construct it from FE. Streaming FE is designed for scenarios where data arrives in a streaming manner and is computed on in an iterative manner as the stream arrives.

First, we define a streaming function to be a stateful function that takes as input a state st_i and a value x_i and outputs the next state st_{i+1} and a value y_i. A streaming FE scheme will compute function keys for streaming functions.

Definition 1.1 (Streaming Function). *A streaming function with state space $\mathcal{S}$, input space $\mathcal{X}$, and output space $\mathcal{Y}$ is a function $f : \mathcal{X} \times \mathcal{S} \to \mathcal{Y} \times \mathcal{S}$.*

- *We define the **output** of f on $x = x_1 \dots x_n \in \mathcal{X}^n$ (denoted $f(x)$) to be $y = y_1 \dots y_n \in \mathcal{Y}^n$ where[1] we have $\mathsf{st}_1 = \perp$ and*

$$(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$$

Definition 1.2 (Streaming Function Class). *The streaming function class $\mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$ is the set of all streaming functions f that have a description $\widehat{f} \in \{0,1\}^{\ell_{\mathcal{F}}}$, state space $\mathcal{S} = \{0,1\}^{\ell_{\mathcal{S}}}$, input space $\mathcal{X} = \{0,1\}^{\ell_{\mathcal{X}}}$, and output space $\mathcal{Y} = \{0,1\}^{\ell_{\mathcal{Y}}}$.*

Now, as we receive the input data $x = x_1 \dots x_n$ in a streaming manner, we would like to be able to encrypt the input and decrypt the streaming function of the encrypted input as it arrives. For encryption, we require the ability to individually generate ciphertexts ct_i for the i^{th} input x_i given only the master public key, x_i, the index i, and an encryption state (which is generated once for x using only the master public key). The decryption algorithm will itself be a streaming function that takes as input the i^{th} ciphertext ct_i, the index i, the function key sk_f, and the current decryption state $\mathsf{Dec.st}_i$ (which roughly speaking encrypts st_i), and outputs the next output value y_i where $(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$ and the next decryption state $\mathsf{Dec.st}_{i+1}$. We now define streaming FE.

Definition 1.3 (Public-Key Streaming FE). *A public-key streaming functional encryption scheme for $\mathsf{P/Poly}$ is a tuple of PPT algorithms $\mathsf{sFE} = (\mathsf{Setup}, \mathsf{EncSetup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ defined as follows:*

[1]We assume that unless specified otherwise, all streaming functions have the same starting state $\perp$ (or the all zero string) which is included in their state space.

- $\mathsf{Setup}(1^\lambda, 1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{S}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}})$*: takes as input the security parameter* λ*, a function size* $\ell_\mathcal{F}$*, a state size* $\ell_\mathcal{S}$*, an input size* $\ell_\mathcal{X}$*, and an output size* $\ell_\mathcal{Y}$*, and outputs the master public key* mpk *and the master secret key* msk*.*

- $\mathsf{EncSetup}(\mathsf{mpk})$*: takes as input the master public key* mpk *and outputs an encryption state* $\mathsf{Enc.st}$[2]*.*

- $\mathsf{Enc}(\mathsf{mpk}, \mathsf{Enc.st}, i, x_i)$*: takes as input the master public key* mpk*, a state* $\mathsf{Enc.st}$*, an index* i*, and a message* $x_i \in \{0,1\}^{\ell_\mathcal{X}}$ *and outputs an encryption* ct_i *of* x_i*.*

- $\mathsf{KeyGen}(\mathsf{msk}, f)$*: takes as input the master secret key* msk *and a function* $f \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$ *and outputs a function key* sk_f*.*

- $\mathsf{Dec}(\mathsf{sk}_f, \mathsf{Dec.st}_i, i, \mathsf{ct}_i)$*: where for each function key* sk_f*,* $\mathsf{Dec}(\mathsf{sk}_f, \cdot, \cdot, \cdot)$ *is a streaming function that takes as input a state* $\mathsf{Dec.st}_i$*, an index* i*, and an encryption* ct_i *and outputs a new state* $\mathsf{Dec.st}_{i+1}$ *and an output* $y_i \in \{0,1\}^{\ell_\mathcal{Y}}$*.*

sFE *satisfies* ***correctness*** *if for all polynomials* p*, there exists a negligible function* μ *such that for all* $\lambda \in \mathbb{N}$*, all* $\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y} \leq p(\lambda)$*, all* $n \in [2^\lambda]$*, all* $x = x_1 \ldots x_n$ *where each* $x_i \in \{0,1\}^{\ell_\mathcal{X}}$*, and all* $f \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$*,*

$$\Pr\left[\overline{\mathsf{Dec}}(\mathsf{sk}_f, \mathsf{ct}_x) = f(x) : \begin{array}{c} (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{S}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}}), \\ \mathsf{ct}_x \leftarrow \overline{\mathsf{Enc}}(\mathsf{mpk}, x) \\ \mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f) \end{array}\right] \geq 1 - \mu(\lambda)$$

where we define[3]

- *the output of* $\overline{\mathsf{Enc}}(\mathsf{mpk}, x)$ *to be* $\mathsf{ct}_x = (\mathsf{ct}_i)_{i\in[n]}$ *produced by sampling* $\mathsf{Enc.st} \leftarrow \mathsf{EncSetup}(\mathsf{mpk})$ *and then computing* $ct_i \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{Enc.st}, i, x_i)$ *for* $i \in [n]$*.*

- *the output of* $\overline{\mathsf{Dec}}(\mathsf{sk}_f, \mathsf{ct}_x)$ *to be* $y = (y_i)_{i\in[n]}$ *where* $(y_i, \mathsf{Dec.st}_{i+1}) = \mathsf{Dec}(\mathsf{sk}_f, \mathsf{Dec.st}_i, i, \mathsf{ct}_i)$

For non-triviality, we require that our streaming FE scheme is *streaming efficient*, meaning that the runtime of our algorithms should not depend on the total length n of the message $x = x_1 \ldots x_n$ that we wish to encrypt. More formally, we require that the size and runtime of all algorithms of sFE on security parameter λ, function size $\ell_\mathcal{F}$, state size $\ell_\mathcal{S}$, input size $\ell_\mathcal{X}$, and output size $\ell_\mathcal{Y}$ are $\mathsf{poly}(\lambda, \ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y})$.

Our security notions are the same as in regular FE except that we allow inputs $x = x_1 \ldots x_n$ of arbitrary length n, allow function keys for streaming functions, and replace $\mathsf{Enc}(\mathsf{mpk}, x)$ with $\overline{\mathsf{Enc}}(\mathsf{mpk}, x)$ as defined in the above definition of streaming FE. In particular, our sFE scheme achieves security similar to standard indistinguishability (IND) security, except that we require all function queries to be made before the challenge message query. This makes our security *function-selective.* However, our definition of security is more liberal than the usual definition of

[2]The purpose of the encryption state is to tie elements of the stream together and must be kept secret in order to prevent mix and match attacks. Suppose that the encryption state either did not exist or was made public. Then an adversary given a function key for some streaming function f and ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_n$ for some stream $x = x_1 \ldots x_n$ could learn the value of f on any extension of the stream. That is, the adversary could encrypt any values $z = z_1 \ldots z_k$ to get ciphertexts $\mathsf{ct}'_1, \ldots, \mathsf{ct}'_k$, and then learn the value of f on any interpolation of x and z (e.g. $x_1z_2z_3x_4z_k$) by simply decrypting using the corresponding interpolation of $\mathsf{ct}_1 \ldots \mathsf{ct}_n$ and $\mathsf{ct}'_1 \ldots \mathsf{ct}'_k$ (e.g. $\mathsf{ct}_1\mathsf{ct}'_2\mathsf{ct}'_3\mathsf{ct}_4\mathsf{ct}'_5$). This is much more power than we want the adversary to have. Note that when the encryption state is secret, then this bad behavior is not allowed as decryption will fail on any ciphertexts ct'_i not generated using the secret encryption state.

[3]As with all streaming functions, we assume that $\mathsf{Dec.st}_1 = \perp$ by default.

function-selective security in that we allow the choice of each function query to depend on the master public key and all previous function queries. For this reason, we say that our scheme achieves *semi-adaptive-function-selective-IND-security* (see Definition 4.6).[4]

We then show how to build sFE from compact FE. Here, compactness means that the runtime of both the setup and encryption algorithms are independent of the function size.[5] This gives us our main theorem.

Theorem 1.4 (Main Theorem)**.** *Assuming a compact, selective-IND-secure, public-key FE scheme for* $\mathsf{P/Poly}$*, there exists a semi-adaptive-function-selective-IND-secure, public-key sFE scheme for* $\mathsf{P/Poly}$*.*

It turns out that the main technical challenge is to build a secret key streaming FE scheme that works even for just one key and one challenge stream to be encrypted. (Please see the Technical Overview for more details.) Our main theorem then follows from this scheme using a bootstrapping approach similar to [AS16].

Additionally, we can build our sFE scheme from well-studied assumptions, and in fact only require polynomial security of these assumptions (unlike the subexponential security needed for $i\mathcal{O}$). Recently, [JLS22] construct sublinear, single-key FE for $\mathsf{P/Poly}$ from well-studied assumptions. We formally define these assumptions in Appendix A.

Theorem 1.5 ([JLS22])**.** *If there exists constants* $\delta, \tau > 0$ *such that:*

- *δ-LPN assumption holds (Definition A.1)*
- *There exists a PRG in* $\mathsf{NC_0}$ *with a stretch of* $n^{1+\tau}$ *where* n *is the length of the input (Definition A.2)*
- *The DLIN assumption over prime order symmetric bilinear groups holds (Definition A.3)*

Then, there exists a sublinear, single-key, selective-IND-secure, public-key FE scheme for $\mathsf{P/Poly}$*.*

[GS16, LM16, AJS15, BV15] show how to bootstrap this to a compact scheme in an unbounded collusion setting with only a polynomial loss in security.

Theorem 1.6 ([GS16, LM16, AJS15, BV15])**.** *If there exists a sublinear, single-key, selective-IND-secure, public-key FE scheme for* $\mathsf{P/Poly}$*, then there exists a compact, selective-IND-secure, public-key FE scheme for* $\mathsf{P/Poly}$*.*

By combining these results, we get the following corollary.

Corollary 1.7. *If there exists constants* $\delta, \tau > 0$ *such that:*

- *δ-LPN assumption holds (Definition A.1)*
- *There exists a PRG in* $\mathsf{NC_0}$ *with a stretch of* $n^{1+\tau}$ *where* n *is the length of the input (Definition A.2)*
- *The DLIN assumption over prime order symmetric bilinear groups holds (Definition A.3)*

Then, there exists a semi-adaptive-function-selective-IND-secure, public-key sFE scheme for $\mathsf{P/Poly}$*.*

[4]We can actually achieve security even when the challenge stream messages depend on the ciphertexts given for the previous stream values. See Remark 4.7.

[5]In other sections of this paper, we refer to this notion as *strong-compactness* since the usual notion of compactness found in the literature only requires the encryption algorithm to be independent of the function size. However, all existing transformations achieving compactness also yield strong compactness.

1.2 Related Work

[AS16] show how to construct FE for Turing machines. While Turing machines internally involve an iterative operation, similar to a streaming function, in contrast to our setting, their final FE scheme still requires the entire input to be known at encryption time and does not produce output until the entire Turing machine computation terminates.

2 Technical Overview

Our goal is to build a public-key sFE scheme for $\mathsf{P/Poly}$. It turns out that it will suffice for us to build a a seemingly weaker primitive: namely a secret-key sFE scheme for $\mathsf{P/Poly}$ that works for just one key and one challenge stream to be encrypted. Our main theorem then follows from this scheme using a bootstrapping approach similar to [AS16]. Thus, we build our scheme in two steps:

1. First, we construct a single-key, single-ciphertext, secret-key sFE scheme $\mathsf{One\text{-}sFE}$. We prove the following:

   Theorem 2.1. *Assuming a strongly-compact, selective-IND-secure, secret-key FE scheme for* $\mathsf{P/Poly}$*, there exists a single-key, single-ciphertext, function-selective-SIM-secure, secret-key sFE scheme for* $\mathsf{P/Poly}$*.*

2. Second, we show how to adapt the technique from [AS16] to bootstrap $\mathsf{One\text{-}sFE}$ into a public-key, sFE scheme sFE. We prove the following:

   Theorem 2.2. *Assuming (1) a selective-IND-secure, public-key FE scheme for* $\mathsf{P/Poly}$*, and (2) a single-key, single-ciphertext, function-selective-IND-secure, secret-key sFE scheme for* $\mathsf{P/Poly}$*, there exists a semi-adaptive-function-selective-IND-secure, public-key sFE scheme for* $\mathsf{P/Poly}$*.*

Together, these two theorems imply our main theorem.

Our main technical contributions is constructing the first single-key, single-ciphertext, secret-key sFE scheme. We then overcome two major obstacles that arise from our construction paradigm.

- Our first approach requires a recursive definition that breaks streaming efficiency by requiring keys of length proportional to the length of the stream.
- A second more subtle issue is that we end up with circular parameter dependencies.

We eliminate both these issues through careful changes to the construction as shown below.

Notation. For notational convenience, in this section, we may omit the security, input size, output size, message size, function size, or state size parameters from our algorithms. Additionally, we will often refer to schemes as being SIM-secure or IND-secure, without specifying whether they are selectively, function-selectively, semi-adaptive-function-selectively, or adaptively secure. We leave these details to the formal proofs.

2.1 Single-Key, Single-Ciphertext, SIM-Secure, Secret-Key Streaming FE

For our first step, we wish to build a secret-key sFE scheme $\mathsf{One\text{-}sFE}$, which is only required to be secure against an adversary who is allowed to make a single function query, followed by a single message query. We will achieve simulation security, meaning that there exists a PPT simulator which can simulate the real function key for f and the real ciphertext for x given only the streaming function f and the output value $y = f(x)$

A Mild Form of SIM-Security for FE. As a warm-up, we first show how to build an *ordinary non-streaming* FE scheme $\mathsf{OneSimFE}$ which achieves a mild form of SIM-security from an IND-secure FE scheme FE and a symmetric key encryption scheme Sym. In particular, our simulation security will only hold against an adversary who is allowed to make a single function query and a single message query. This mild form of simulation security will be useful in building streaming FE, and we will use this technique throughout this section.

- $\mathsf{OneSimFE.Setup}(1^\lambda)$:
    1. $\mathsf{msk} \leftarrow \mathsf{FE.Setup}(1^\lambda)$
    2. $k \leftarrow \mathsf{Sym.Setup}(1^\lambda)$
    3. Output $(\mathsf{mpk}' = \mathsf{mpk}, \mathsf{msk}' = (\mathsf{msk}, k))$
- $\mathsf{OneSimFE.Enc}(\mathsf{mpk}, x) = \mathsf{FE.Enc}(\mathsf{mpk}, (x, 0, \perp, \perp))$
- $\mathsf{OneSimFE.KeyGen}((\mathsf{msk}, k), f)$:
    1. $c \leftarrow \mathsf{Sym.Enc}(k, 0)$
    2. Output $\mathsf{FE.KeyGen}(\mathsf{msk}, g_{f,c})$ where we define

```
$g_{f,c}(x, \alpha, k, v)$ :
  – If $\alpha = 0$, output $f(x)$.                 //$\alpha = 0$ is the “normal” case.
  – Else, output $v \oplus \mathsf{Sym.Dec}(k, c)$.    // This branch is for simulation.
```

- $\mathsf{OneSimFE.Dec}(\mathsf{sk}_f, \mathsf{ct}) = \mathsf{FE.Dec}(\mathsf{sk}_f, \mathsf{ct})$

In our simulation security game, there are two cases:

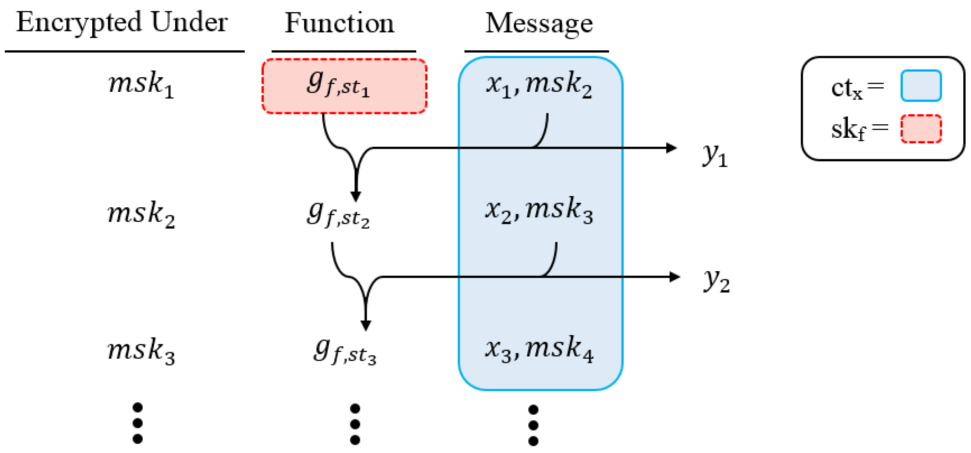
- **Case 1: The message query x is asked before the function query f.**
  On receiving a message query length n, the simulator Sim outputs a simulated ciphertext $\mathsf{ct} \leftarrow \mathsf{FE.Enc}(\mathsf{mpk}, (0^n, 1, k, 0))$. On receiving a function query f along with $f(x)$, Sim outputs a simulated function key $\mathsf{sk}_f \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}, g_{f,c'})$ where $c' \leftarrow \mathsf{Sym.Enc}(k, f(x))$.
- **Case 2: The function query f is asked before the message query x.**
  On receiving a function query f, the simulator Sim outputs a simulated function key $\mathsf{sk}_f \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}, g_{f,c})$ where $c \leftarrow \mathsf{Sym.Enc}(k, 0)$. On receiving a message query length n and $f(x)$, Sim outputs a simulated ciphertext $\mathsf{ct} \leftarrow \mathsf{FE.Enc}(\mathsf{mpk}, (0^n, 1, k, f(x))$.

Simulation security then follows by the IND-security of FE since $g_{f,c}(x, 0, \perp, \perp) = g_{f,c}(0^n, 1, k, f(x)) = g_{f,c'}(x, 0, \perp, \perp) = g_{f,c'}(0^n, 1, k, 0) = f(x)$ and $c \approx c'$ by the security of Sym.

With this simple initial tool in our belt, we now proceed to tackle the main problem – building streaming FE.

First Attempt at Building $\mathsf{One\text{-}sFE}$. Each iteration of our streaming FE scheme needs to combine two values: the current input x_i and the current state st_i. Our first observation is that regular FE allows us to securely combine two values: a function and an input. Thus, if we were to place x_i in a FE ciphertext and place st_i (and f) in a corresponding FE function key, then we could hope to use FE to securely combine the two values and compute $f(x_i, \mathsf{st}_i)$. Now, $\mathsf{st}_1 = \perp$ is fixed and known at key generation time. Thus, we can generate the first function key containing f and

st_1. But how do we generate keys containing future states? Our main intuition here is to have the function key containing st_i and f not only compute $f(x_i, \mathsf{st}_i)$ and output y_i, but also create the next function key for the next state st_{i+1}. This gives us the following initial construction: The ciphertext for x is $\mathsf{ct}_x = \{\mathsf{ct}_i\}_{i\in[n]}$ where each $\mathsf{ct}_i \leftarrow \mathsf{OneSimFE.Enc}(\mathsf{msk}_i, (x_i, \mathsf{msk}_{i+1}))$. The function key for f is $\mathsf{sk}_f = \mathsf{sk}_{g_{f,\mathsf{st}_1}} \leftarrow \mathsf{OneSimFE.KeyGen}(\mathsf{msk}_1, g_{f,\mathsf{st}_1})$ for g_{f,st_1} as defined below. Here, we use a different master secret key msk_i for each iteration i as our simulation security technique only allows us to program a single value into each ciphertext or key. We can generate all of the $\mathsf{One\text{-}sFE}$ master secret keys $\{\mathsf{msk}_i\}$ from a short PRF key, which will be the master secret key of our streaming FE scheme. The diagram below depicts how we can combine ct_x and sk_f to learn $f(x)$.



$g_{f,\mathsf{st}_i}(x_i, \mathsf{msk}_{i+1})$:

1. $(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$
2. Output $(y_i, \mathsf{OneSimFE.KeyGen}(\mathsf{msk}_{i+1}, g_{f,\mathsf{st}_{i+1}}))$

Figure 1: First attempt at building $\mathsf{One\text{-}sFE}$.

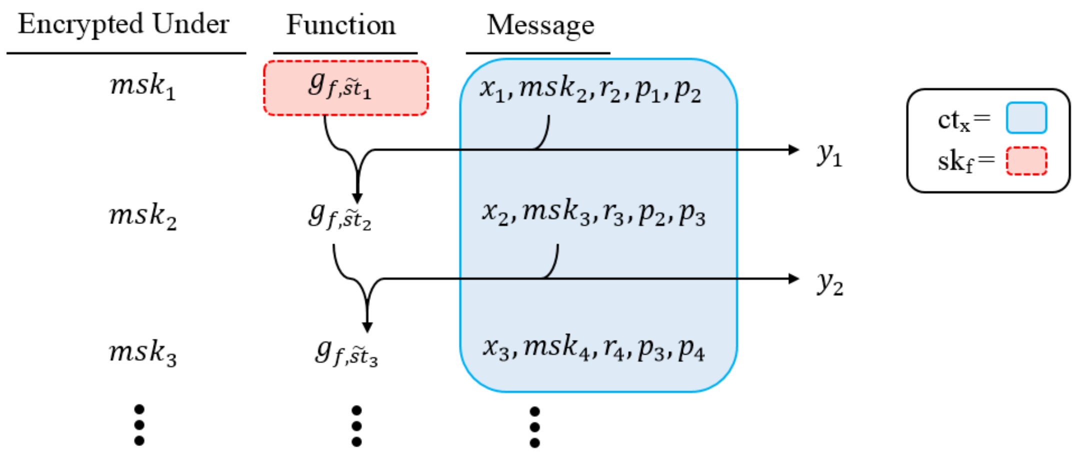
The idea behind this attempt is that we want to prove security by one by one replacing each $(\mathsf{sk}_{g_{f,\mathsf{st}_i}}, \mathsf{ct}_i)$ with simulated values using the security of $\mathsf{OneSimFE}$. Observe, that simulating $(\mathsf{sk}_{g_{f,\mathsf{st}_i}}, \mathsf{ct}_i)$ removes msk_{i+1} from ct_i, hopefully allowing us to then simulate the next $(\mathsf{sk}_{g_{f,\mathsf{st}_{i+1}}}, \mathsf{ct}_{i+1})$ (as msk_{i+1} is hidden). Unfortunately, this initial scheme does not work and has three main issues:

1. $\mathsf{OneSimFE}$ only creates function keys for deterministic functions, but $\mathsf{OneSimFE.KeyGen}$ (and thus each g_{f,st_i}) is a randomized function.

2. As $\mathsf{OneSimFE}$ is not function-hiding, the value of each intermediate st_i is made public, thus compromising security. (In particular, simulating $(\mathsf{sk}_{g_{f,\mathsf{st}_i}}, \mathsf{ct}_i)$ requires us to know the output value $(y_i, \mathsf{sk}_{g_{f,\mathsf{st}_{i+1}}})$.)

3. The definition of g_{f,st_i} is recursive, and thus the size of our initial function key g_{f,st_1} will depend on the total number n of recursive steps we wish to take. Therefore, our scheme is not streaming efficient as it depends on the length of x.

Solving the randomization and state privacy issues. We can easily fix the first two issues. We can make g_{f,st_i} deterministic by simply giving the randomness r_i needed to compute $\mathsf{OneSimFE.KeyGen}$ as input to g_{f,st_i} by placing this randomness in the i^{th} ciphertext. To fix the second issue, instead of giving out function keys with st_i hardcoded into them, we give out function keys with $\widetilde{\mathsf{st}}_i$ hardcoded into them where $\widetilde{\mathsf{st}}_i = \mathsf{st}_i \oplus p_i$ for a random pad p_i. We then simply add p_i and p_{i+1} into the ciphertext for x_i so that we can pad and un-pad states st_i and st_{i+1} in the i^{th} iteration. As each $\widetilde{\mathsf{st}}_i$ is uniformly random when pad p_i is hidden, then giving out $\widetilde{\mathsf{st}}_i$ should not compromise security since p_i is hidden in the ciphertext. This gives us the following intermediate scheme. The ciphertext for x is $\mathsf{ct}_x = \{\mathsf{ct}_i\}_{i\in[n]}$ where each $\mathsf{ct}_i \leftarrow \mathsf{OneSimFE.Enc}(\mathsf{msk}_i, (x_i, \mathsf{msk}_{i+1}, r_{i+1}, p_i, p_{i+1}))$. The function key for f is $\mathsf{sk}_f = \mathsf{sk}_{g_{f,\widetilde{\mathsf{st}}_1}} \leftarrow \mathsf{OneSimFE.KeyGen}(\mathsf{msk}_1, g_{f,\widetilde{\mathsf{st}}_1})$ where $\widetilde{\mathsf{st}}_1 = \mathsf{st}_1 \oplus p_1$ and $g_{f,\widetilde{\mathsf{st}}_1}$ is defined as below. We can generate all of the $\mathsf{OneSimFE}$ master secret keys $\{\mathsf{msk}_i\}$ and the pads $\{p_i\}$ from a short PRF key, which will be the master secret key of our streaming FE scheme. The diagram below depicts how we can combine ct_x and sk_f to learn $f(x)$.



$g_{f,\widetilde{\mathsf{st}}_i}(x_i, \mathsf{msk}_{i+1}, r_{i+1}, p_i, p_{i+1})$:

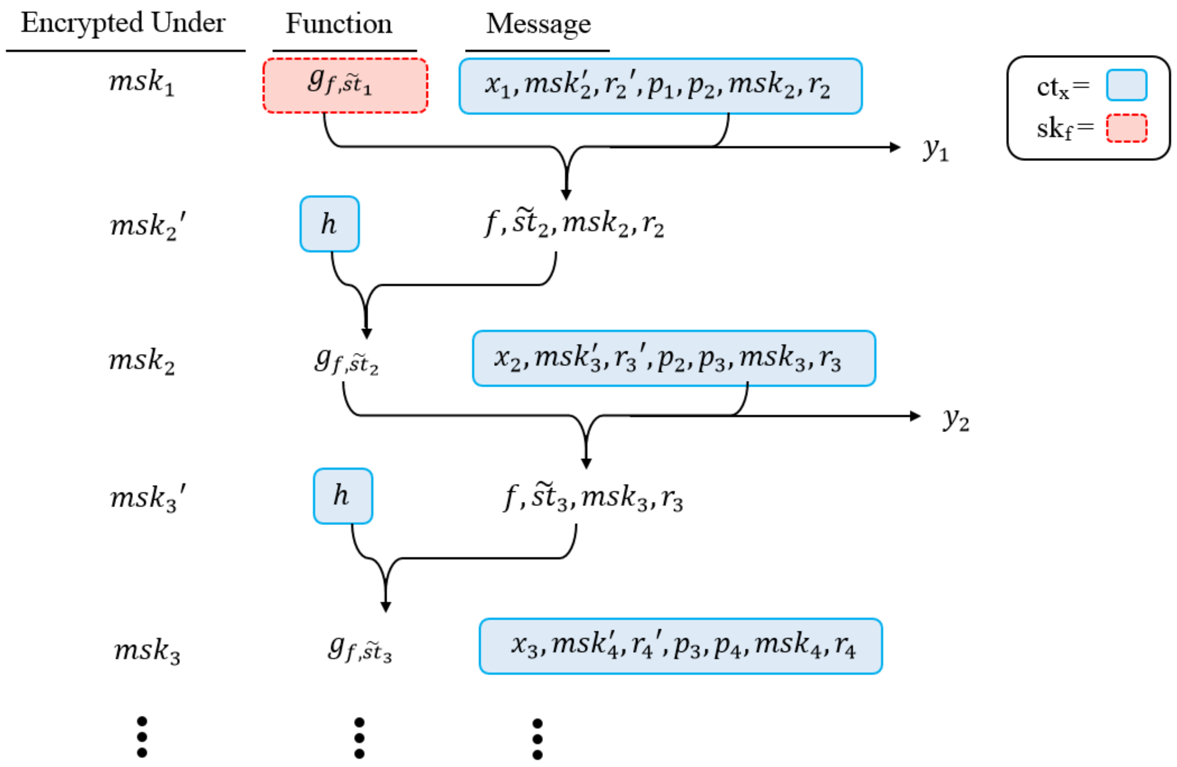
1. $\mathsf{st}_i = \widetilde{\mathsf{st}}_i \oplus p_i$
2. $(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$
3. $\widetilde{\mathsf{st}}_{i+1} = \mathsf{st}_{i+1} \oplus p_{i+1}$
4. Output $(y_i, \mathsf{OneSimFE.KeyGen}(\mathsf{msk}_{i+1}, g_{f,\widetilde{\mathsf{st}}_{i+1}}; r_{i+1}))$

Figure 2: Solving the randomization and state privacy issues.

Again, the definition of g_{f,st_i} is recursive, so this scheme is not streaming efficient. Indeed, achieving streaming efficiency, where the complexity of each encryption and decryption do not grow with n, is the main technical barrier we need to overcome.

Achieving Streaming Efficiency, Part 1: Removing the Recursive Definition. To fix the issue of the recursive definition, we split each $g_{f,\widetilde{\mathsf{st}}_i}$ into two functions. Rather than having $g_{f,\widetilde{\mathsf{st}}_i}$

generate the function key for $g_{f,\widetilde{\mathsf{st}}_{i+1}}$, we have $g_{f,\widetilde{\mathsf{st}}_i}$ simply generate an encryption of f and $\widetilde{\mathsf{st}}_i$, and have a different function h generate $g_{f,\widetilde{\mathsf{st}}_{i+1}}$ from f and $\widetilde{\mathsf{st}}_i$. This gives us the following scheme.



Encrypted Under
Function
Message
msk_1
$g_{f,\widetilde{st}_1}$
$x_1, msk_2', r_2', p_1, p_2, msk_2, r_2$
y_1
$ct_x=$
$sk_f=$
msk_2'
h
$f, \widetilde{st}_2, msk_2, r_2$
msk_2
$g_{f,\widetilde{st}_2}$
$x_2, msk_3', r_3', p_2, p_3, msk_3, r_3$
y_2
msk_3'
h
$f, \widetilde{st}_3, msk_3, r_3$
msk_3
$g_{f,\widetilde{st}_3}$
$x_3, msk_4', r_4', p_3, p_4, msk_4, r_4$

```
$g_{f,\widetilde{\mathsf{st}}_i}(x_i, \mathsf{msk}'_{i+1}, r'_{i+1}, p_i, p_{i+1}, \mathsf{msk}_{i+1}, r_{i+1})$:
    1. $\mathsf{st}_i = \widetilde{\mathsf{st}}_i \oplus p_i$
    2. $(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$
    3. $\widetilde{\mathsf{st}}_{i+1} = \mathsf{st}_{i+1} \oplus p_{i+1}$
    4. Output $(y_i, \mathsf{OneSimFE.Enc}(\mathsf{msk}'_{i+1}, (f, \widetilde{\mathsf{st}}_{i+1}, \mathsf{msk}_{i+1}, r_{i+1}); r'_{i+1}))$
```

```
$h(f, \widetilde{\mathsf{st}}_{i+1}, \mathsf{msk}_{i+1}, r_{i+1})$:
    1. Output $\mathsf{OneSimFE.KeyGen}(\mathsf{msk}_{i+1}, g_{f,\widetilde{\mathsf{st}}_{i+1}}; r_{i+1})$
```

Figure 3: Removing the recursive definition.

The ciphertext for x is $\mathsf{ct}_x = \{\mathsf{ct}_i, \mathsf{sk}_{h_i}\}_{i\in[n]}$ where each $\mathsf{ct}_i \leftarrow \mathsf{OneSimFE.Enc}(\mathsf{msk}_i, (x_i, \mathsf{msk}'_{i+1}, r'_{i+1}, p_i, p_{i+1}, \mathsf{msk}_{i+1}, r_{i+1}))$ and $\mathsf{sk}_{h_i} \leftarrow \mathsf{OneSimFE.KeyGen}(\mathsf{msk}'_i, h)$ for h defined below. The function key for f is $\mathsf{sk}_f = \mathsf{sk}_{g_{f,\widetilde{\mathsf{st}}_1}} \leftarrow \mathsf{OneSimFE.KeyGen}(\mathsf{msk}_1, g_{f,\widetilde{\mathsf{st}}_1})$ where $\widetilde{\mathsf{st}}_1 = \mathsf{st}_1 \oplus p_1$ and $g_{f,\widetilde{\mathsf{st}}_1}$ is defined below. We can generate all of the One-sFE master secret keys $\{\mathsf{msk}_i, \mathsf{msk}'_i\}$ and the pads $\{p_i\}$ from a short PRF key, which will be the master secret key of our streaming FE scheme. The diagram above depicts how we can combine ct_x and sk_f to learn $f(x)$.

Unfortunately, although the definitions of $g_{f,\widetilde{\mathsf{st}}_i}$ (and h) are no longer recursive, the scheme written here has circularly-dependent parameters. In particular, $\mathsf{OneSimFE}$ must generate function keys for its own key generation and encryption algorithms as it must generate function keys for h (which contains $\mathsf{OneSimFE.KeyGen}$) and function keys for $g_{f,\widetilde{\mathsf{st}}_i}$, (which contains $\mathsf{OneSimFE.Enc}$).

Achieving Streaming Efficiency, Part 2: Fixing the Circular Dependencies. To remove the circular dependencies among the parameters, we will make two changes:

- Rather than encrypting msk'_i and msk_i in our ciphertexts, we will instead encrypt the randomness r'_{msk_i} and r_{msk_i} used to generate these values. We can then generate $\mathsf{msk}'_i, \mathsf{msk}_i$ from this randomness within $g_{f,\widetilde{\mathsf{st}}_i}$ and h by using the setup algorithm. This will allow us to bound the size of our FE messages as we can assume without loss of generality that the size of all randomness used is λ (if we need additional randomness, our algorithms can simply expand this randomness using a PRG).

- We will use two different FE schemes: one scheme $\mathsf{OneSimFE}$ for $g_{f,\widetilde{\mathsf{st}}_i}$, and the other scheme $\mathsf{OneSimFE}'$ for h. Additionally, we will require that $\mathsf{OneSimFE}'$ is strongly-compact, meaning that the setup and encryption algorithms do not depend on the function size and output size.

Now we can instantiate our parameters.

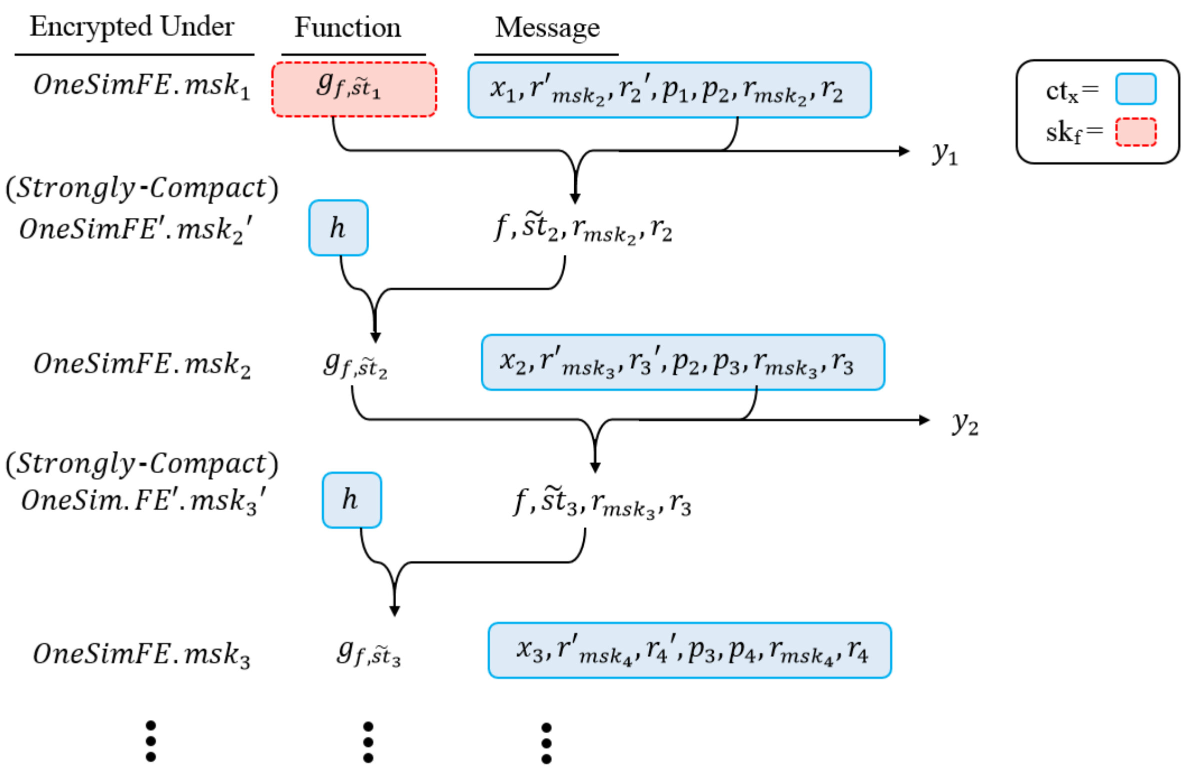
1. Since we are encrypting $r_{\mathsf{msk}_i}, r'_{\mathsf{msk}_i}$ instead of $\mathsf{msk}_i, \mathsf{msk}'_i$, we can bound the size of the inputs to both $\mathsf{OneSimFE}$ and $\mathsf{OneSimFE}'$.

2. Since we know the input size of $\mathsf{OneSimFE}'$, by the strong-compactness of $\mathsf{OneSimFE}'$, we can determine the sizes of $\mathsf{OneSimFE}'.\mathsf{Setup}$ and $\mathsf{OneSimFE}'.\mathsf{Enc}$ and thus of $g_{f,\widetilde{\mathsf{st}}_i}$.

3. Since we know the function size (i.e. the size of $g_{f,\widetilde{\mathsf{st}}_i}$), input size, and output size of functions of $\mathsf{OneSimFE}$, this allows us to determine the parameters of $\mathsf{OneSimFE}$. Thus, we can determine the sizes of $\mathsf{OneSimFE.Setup}$ and $\mathsf{OneSimFE.KeyGen}$ and therefore of h.

4. Finally, this allows us to determine the parameters of $\mathsf{OneSimFE}'$ which generates keys for h.

Now, we have the following scheme. The ciphertext for x is $\mathsf{ct}_x = \{\mathsf{ct}_i, \mathsf{sk}_{h_i}\}_{i \in [n]}$ where each $\mathsf{ct}_i \leftarrow \mathsf{OneSimFE.Enc}(\mathsf{msk}_i, (x_i, r'_{\mathsf{msk}_{i+1}}, r'_{i+1}, p_i, p_{i+1}, r_{\mathsf{msk}_{i+1}}, r_{i+1}))$ and $\mathsf{sk}_{h_i} \leftarrow \mathsf{OneSimFE}'.\mathsf{KeyGen}(\mathsf{msk}'_i, h)$ for h defined below. The function key for f is $\mathsf{sk}_f = \mathsf{sk}_{g_{f,\widetilde{\mathsf{st}}_1}} \leftarrow \mathsf{OneSimFE.KeyGen}(\mathsf{msk}_1, g_{f,\widetilde{\mathsf{st}}_1})$ where $\widetilde{\mathsf{st}}_1 = \mathsf{st}_1 \oplus p_1$ and $g_{f,\widetilde{\mathsf{st}}_1}$ is defined below. We can generate all of the $\mathsf{One\text{-}sFE}$ master secret keys $\{\mathsf{msk}_i, \mathsf{msk}'_i\}$, the randomness $\{r_{\mathsf{msk}_i}, r_{\mathsf{msk}'_i}\}$ needed to compute them, and the pads $\{p_i\}$ from a short PRF key, which will be the master secret key of our streaming FE scheme. The diagram below depicts how we can combine ct_x and sk_f to learn $f(x)$.



Encrypted Under
Function
Message
$OneSimFE.msk_1$
$g_{f,\widetilde{st}_1}$
$x_1, r'_{msk_2}, r_2', p_1, p_2, r_{msk_2}, r_2$
y_1
$ct_x =$
$sk_f =$
(Strongly-Compact) $OneSimFE'.msk_2'$
h
$f, \widetilde{st}_2, r_{msk_2}, r_2$
$OneSimFE.msk_2$
$g_{f,\widetilde{st}_2}$
$x_2, r'_{msk_3}, r_3', p_2, p_3, r_{msk_3}, r_3$
y_2
(Strongly-Compact) $OneSim.FE'.msk_3'$
h
$f, \widetilde{st}_3, r_{msk_3}, r_3$
$OneSimFE.msk_3$
$g_{f,\widetilde{st}_3}$
$x_3, r'_{msk_4}, r_4', p_3, p_4, r_{msk_4}, r_4$

```
$g_{f,\widetilde{\mathsf{st}}_i}(x_i, r'_{\mathsf{msk}_{i+1}}, r'_{i+1}, p_i, p_{i+1}, r_{\mathsf{msk}_{i+1}}, r_{i+1})$:
    1. $\mathsf{st}_i = \widetilde{\mathsf{st}}_i \oplus p_i$
    2. $(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$
    3. $\widetilde{\mathsf{st}}_{i+1} = \mathsf{st}_{i+1} \oplus p_{i+1}$
    4. $\mathsf{msk}'_{i+1} \leftarrow \mathsf{OneSimFE'.Setup}(1^\lambda; r'_{\mathsf{msk}_{i+1}})$
    5. Output $(y_i, \mathsf{OneSimFE'.Enc}(\mathsf{msk}'_{i+1}, (f, \widetilde{\mathsf{st}}_{i+1}, r_{\mathsf{msk}_{i+1}}, r_{i+1}); r'_{i+1}))$
```

```
$h(f, \widetilde{\mathsf{st}}_{i+1}, r_{\mathsf{msk}_{i+1}}, r_{i+1})$:
    1. $\mathsf{msk}_{i+1} \leftarrow \mathsf{OneSimFE.Setup}(1^\lambda; r_{\mathsf{msk}_{i+1}})$
    2. Output $\mathsf{OneSimFE.KeyGen}(\mathsf{msk}_{i+1}, g_{f,\widetilde{\mathsf{st}}_{i+1}}; r_{i+1})$
```

Figure 4: Fixing the circular dependencies.

To prove security, we will iteratively replace each ciphertext and function key with simulated values.

1. First, we use the SIM-security of OneSimFE to replace ct_1 and $\mathsf{sk}_{g_{f,\widetilde{\mathsf{st}}_1}}$ with simulated values.

The simulation only requires knowledge of the function $g_{f,\widetilde{\mathsf{st}}_1}$ and the output values y_1 and $\mathsf{ct}'_2 = \mathsf{OneSimFE}'.\mathsf{Enc}(\mathsf{msk}'_2, (f, \widetilde{\mathsf{st}}_2, r_{\mathsf{msk}_2}, r_2); r'_2)$. This change removes the values of x_1 and p_1 from the experiment, which ensures that $\widetilde{\mathsf{st}}_1$ can be made uniformly random and does not leak any information. Additionally, msk'_2 (and r'_{msk_2}) are now only used to generate ct'_2 and sk_{h_2}.

2. Next, we use the SIM-security of $\mathsf{OneSimFE}'$ to replace ct'_2 and sk_{h_2} with simulated values. The simulation only requires knowledge of the function h and the output value $\mathsf{sk}_{g_{f,\widetilde{\mathsf{st}}_2}} \leftarrow \mathsf{OneSimFE}.\mathsf{KeyGen}(\mathsf{msk}_2, g_{f,\widetilde{\mathsf{st}}_2}; r_2)$. Now, msk_2 (and r_{msk_2}) are only used to generate ct_2 and $\mathsf{sk}_{g_{f,\widetilde{\mathsf{st}}_2}}$.

3. As in step 1, we replace ct_2 and $\mathsf{sk}_{g_{f,\widetilde{\mathsf{st}}_2}}$ with simulated values. This hides x_2, msk'_3, and r'_{msk_3}.

4. As in step 2, we replace ct'_2 and sk_{h_2} with simulated values. This hides msk_3 and r_{msk_3}.

5. We then repeat steps 3 and 4 in order for every $(\mathsf{ct}_i, \mathsf{sk}_{g_{f,\widetilde{\mathsf{st}}_i}})$ and $(\mathsf{ct}'_i, \mathsf{sk}_{h_i})$.

Once all ciphertexts and function keys have been simulated, then we are in an ideal world, simulator experiment. Thus, we achieve single-key, single-ciphertext, SIM-security, as long as the challenge function f is given before the challenge message x. This is because in order to simulate each sk_{h_i} in the i^{th} ciphertext for x, we must know the output value $\mathsf{sk}_{g_{f,\widetilde{\mathsf{st}}_i}}$ and thus must know f.

Final Scheme. Our final scheme is the same as the previous construction except that we instantiate $\mathsf{OneSimFE}$ and $\mathsf{OneSimFE}'$ from standard FE, using techniques similar to the one described at the beginning of this technical overview. This requires a little care to ensure that we do not introduce new circular dependencies.

2.2 Bootstrapping to an IND-Secure, Public-Key Streaming FE

Here, we use the same technique that was used in [AS16] to bootstrap a single-key, single-ciphertext FE scheme for Turing machines into a public-key FE scheme for Turing machine. Our construction is nearly the same as in [AS16], with only a few minor modifications (see Remark 6.5). Thus, we will only provide an abbreviated overview of this technique.

Let FE be a selective-IND-secure, public-key FE scheme. Let FPFE be a function-private-selective-IND-secure, secret-key FE scheme. (This can be built from FE using techniques from [BS18].) Let $\mathsf{One\text{-}sFE}$ be our single-key, single-ciphertext, function-selective-SIM-secure, secret-key streaming FE scheme. Let PRF and $\mathsf{PRF2}$ be secure PRFs.

At a high level, the idea is to generate a new $\mathsf{One\text{-}sFE}$ master secret key $\mathsf{One\text{-}sFE.msk}$ for each message x and function f. This ensures that each $\mathsf{One\text{-}sFE.msk}$ is only used for one key and one ciphertext, allowing us to then rely on the security of $\mathsf{One\text{-}sFE}$. This is implemented in two steps:

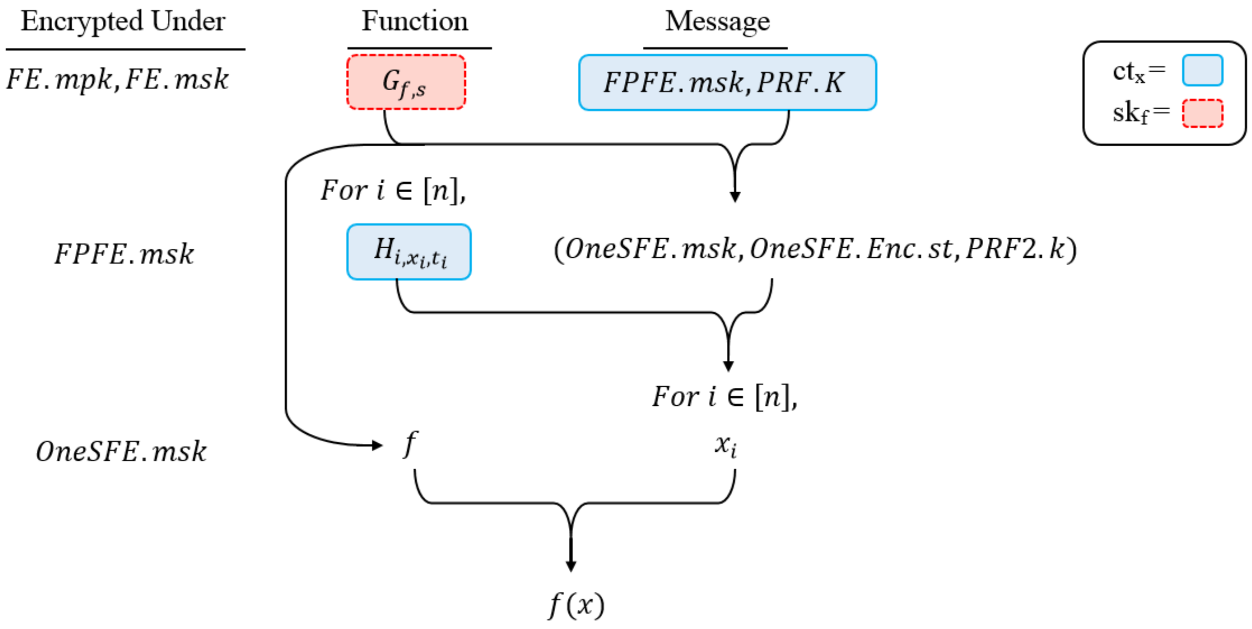
1. First, we use FE to combine a PRF key $\mathsf{PRF}.k$ from the ciphertext for x with randomness s from the function key for f to securely generate a fresh $\mathsf{One\text{-}sFE.msk}$ for (x, f). We then use $\mathsf{One\text{-}sFE.msk}$ to generate a function key $\mathsf{One\text{-}sFE.sk}_f$ for f and a ciphertext $\mathsf{FPFE.ct}$ encrypting $\mathsf{One\text{-}sFE.msk}$.

2. Second, our ciphertext for x creates FPFE function keys with values from x hardcoded into them. The function privacy of FPFE will ensure that this does not leak information about x. These function keys can then be combined with $\mathsf{FPFE.ct}$ to get an encryption $\mathsf{One\text{-}sFE.ct}_x$ of x.

This gives us the following scheme, which is close to our actual construction.[6] The ciphertext for $x = x_1 \ldots x_n$ is $\mathsf{ct}_x = (\mathsf{FE.ct}, \{\mathsf{FPFE.sk}_{H_{i,x_i,t_i}}\}_{i \in [n]})$ where $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (\mathsf{FPFE.msk}, \mathsf{PRF}.K))$ and $\mathsf{FPFE.sk}_{H_{i,x_i,t_i}} \leftarrow \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H_{i,x_i,t_i})$ for H_{i,x_i,t_i} defined below and a random t_i. The function key for f is $\mathsf{sk}_f = \mathsf{FE.sk}_{G_{f,s}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_{f,s})$ for $G_{f,s}$ defined below. The diagram below depicts how we can combine ct_x and sk_f to learn $f(x)$.

To prove security, we will first use a similar simulation technique as in our One-sFE construction to ensure that each One-sFE.msk is securely generated. This is done by programming into each $G_{f,s}$ the output value $(\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, \mathsf{PRF2}.k)$ generated by $G_{f,s}(\mathsf{FPFE.msk}, \mathsf{PRF}.K)$. Next, we will move from encrypting $x^{(b)}$ for a random $b \leftarrow \{0,1\}$ to always encrypting $x^{(0)}$. This will prove security as our final hybrid will be independent of b. We will perform this change from $x^{(b)}$ to $x^{(0)}$ one function at a time by utilizing the security of One-sFE and FPFE to switch between different branches of computation within H_{i,x_i,t_i} (which we add into H_{i,x_i,t_i} using the function privacy of FPFE). We leave further details to the formal proof.

[6]Our actual scheme adds additional branches of computation to $G_{f,s}$ and H_{i,x_i,t_i} which are only used in the security proof.



```
$G_{f,s}(\mathsf{FPFE.msk}, \mathsf{PRF}.K)$:
    1. $(r_{\mathsf{Setup}}, r_{\mathsf{KeyGen}}, r_{\mathsf{EncSetup}}, r_{\mathsf{PRF2}}, r_{\mathsf{Enc}}) \leftarrow \mathsf{PRF.Eval}(\mathsf{PRF}.K, s)$
    2. $\mathsf{One\text{-}sFE.msk} \leftarrow \mathsf{One\text{-}sFE.Setup}(1^\lambda; r_{\mathsf{Setup}})$
    3. $\mathsf{One\text{-}sFE.Enc.st} \leftarrow \mathsf{One\text{-}sFE.EncSetup}(\mathsf{One\text{-}sFE.msk}; r_{\mathsf{EncSetup}})$
    4. $\mathsf{One\text{-}sFE.sk}_f \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}, f; r_{\mathsf{KeyGen}})$
    5. $\mathsf{PRF2}.k \leftarrow \mathsf{PRF2.Setup}(1^\lambda; r_{\mathsf{PRF2}})$
    6. $\mathsf{FPFE.ct} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, \mathsf{PRF2}.k); r_{\mathsf{Enc}})$
    7. Output $(\mathsf{One\text{-}sFE.sk}_f, \mathsf{FPFE.ct})$
```

```
$H_{i,x_i,t_i}(\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, \mathsf{PRF2}.k)$:
    1. $r_i \leftarrow \mathsf{PRF2.Eval}(\mathsf{PRF2}.k, t_i)$
    2. Output $\mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, i, x_i; r_i)$
```

Figure 5: Bootstrapping to an IND-secure, public-key streaming FE. This is similar, but not identical to our final construction.

3 Preliminaries

Throughout, we will use λ to denote a security parameter.

Notation.

- We say that a function $f(\lambda)$ is negligible in λ if $f(\lambda) = \lambda^{-\omega(1)}$, and we denote it by $f(\lambda) = \mathsf{negl}(\lambda)$.
- We say that a function $g(\lambda)$ is polynomial in λ if $g(\lambda) = p(\lambda)$ for some fixed polynomial p, and we denote it by $g(\lambda) = \mathsf{poly}(\lambda)$.
- For $n \in \mathbb{N}$, we use $[n]$ to denote $\{1, \ldots, n\}$.
- If R is a random variable, then $r \leftarrow R$ denotes sampling r from R. If T is a set, then $i \leftarrow T$ denotes sampling i uniformly at random from T.

We will use PRFs and symmetric key encryption schemes with pseudorandom ciphertexts. We formally define these notions in Appendix B.1.

3.1 Functional Encryption

Here we give some fundamental definitions for functional encryption (FE) schemes. First, we define a class of functions parameterized by function size, input length, and output length.

Definition 3.1 (Function Class)**.** *The function class $\mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$ is the set of all functions f that have a description $\widehat{f} \in \{0,1\}^{\ell_{\mathcal{F}}}$, take inputs in $\{0,1\}^{\ell_{\mathcal{X}}}$, and output values in $\{0,1\}^{\ell_{\mathcal{Y}}}$.*

3.1.1 Public-Key Functional Encryption

Definition 3.2 (Public-Key Functional Encryption)**.** *A public-key functional encryption scheme for* $\mathsf{P/Poly}$ *is a tuple of PPT algorithms* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *defined as follows:*[7]

- $\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}})$*: takes as input the security parameter λ, a function size $\ell_{\mathcal{F}}$, an input size $\ell_{\mathcal{X}}$, and an output size $\ell_{\mathcal{Y}}$, and outputs the master public key* mpk *and the master secret key* msk.
- $\mathsf{Enc}(\mathsf{mpk}, x)$*: takes as input the master public key* mpk *and a message $x \in \{0,1\}^{\ell_{\mathcal{X}}}$, and outputs an encryption* ct *of x.*
- $\mathsf{KeyGen}(\mathsf{msk}, f)$*: takes as input the master secret key* msk *and a function $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$, and outputs a function key* sk_f.
- $\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct})$*: takes as input a function key* sk_f *and a ciphertext* ct*, and outputs a value $y \in \{0,1\}^{\ell_{\mathcal{Y}}}$.*

FE *satisfies* ***correctness*** *if for all polynomials p, there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$, all $\ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}} \leq p(\lambda)$, all $x \in \{0,1\}^{\ell_{\mathcal{X}}}$, and all $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$,*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}_x) = f(x) : \begin{array}{c} (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}}) \\ \mathsf{ct}_x \leftarrow \mathsf{Enc}(\mathsf{mpk}, x) \\ \mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f) \end{array}\right] \geq 1 - \mu(\lambda).$$

[7] We also allow $\mathsf{Enc}, \mathsf{KeyGen}$, and Dec to additionally receive parameters $1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}}$ as input, but omit them from our notation for convenience.

There are many definitions of security. We define only a few here. Selective-IND-security requires the challenge message to be sent first.

Definition 3.3 (Selective-IND-Security)**.** *A public-key functional encryption scheme* FE *for* $\mathsf{P/Poly}$ *is selective-IND-secure if there exists a negligible function* μ *such that for all* $\lambda \in \mathbb{N}$ *and every PPT adversary* $\mathcal{A}$*,*

$$\left|\Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{Sel\text{-}IND}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{Sel\text{-}IND}}(1^\lambda, 1) = 1]\right| \leq \mu(\lambda)$$

where for each $b \in \{0,1\}$ *and* $\lambda \in \mathbb{N}$*, we define*

$\mathsf{Expt}_{\mathcal{A}}^{\mathsf{Sel\text{-}IND}}(1^\lambda, b)$

1. ***Parameters****:* $\mathcal{A}$ *takes as input* 1^λ*, and outputs a function size* $1^{\ell_{\mathcal{F}}}$*, an input size* $1^{\ell_{\mathcal{X}}}$*, and an output size* $1^{\ell_{\mathcal{Y}}}$*.*
2. ***Challenge Message****:* $\mathcal{A}$ *outputs a challenge message pair* (x_0, x_1) *where* $x_0, x_1 \in \{0,1\}^{\ell_{\mathcal{X}}}$*.*
3. ***Public Key and Challenge Ciphertext****:*
   (a) $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}})$
   (b) $\mathsf{ct} \leftarrow \mathsf{FE.Enc}(\mathsf{mpk}, x_b)$
   (c) *Send* $(\mathsf{mpk}, \mathsf{ct})$ *to* $\mathcal{A}$*.*
4. ***Function Queries****: The following can be repeated any polynomial number of times:*
   (a) $\mathcal{A}$ *outputs a function query* $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$
   (b) $\mathsf{sk}_f \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}, f)$
   (c) *Send* sk_f *to* $\mathcal{A}$
5. ***Experiment Outcome****:* $\mathcal{A}$ *outputs a bit* b'*. The output of the experiment is set to* 1 *if* $b = b'$ *and* $f(x_0) = f(x_1)$ *for all functions* f *queried by the adversary.*

Semi-adaptive-function-selective-IND-security allows the adversary to receive the master public key at the start of the experiment, but requires the adversary to specify all function queries before receiving the challenge message.

Definition 3.4 (Semi-Adaptive-Function-Selective-IND-Security)**.** *A public-key functional encryption scheme* FE *for* $\mathsf{P/Poly}$ *is semi-adaptive-function-selective-IND-secure if there exists a negligible function* μ *such that for all* $\lambda \in \mathbb{N}$ *and every PPT adversary* $\mathcal{A}$*,*

$$\left|\Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{Semi\text{-}Ad\text{-}Func\text{-}Sel\text{-}IND}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{Semi\text{-}Ad\text{-}Func\text{-}Sel\text{-}IND}}(1^\lambda, 1) = 1]\right| \leq \mu(\lambda)$$

where for each $b \in \{0,1\}$ *and* $\lambda \in \mathbb{N}$*, we define*

$\mathsf{Expt}_{\mathcal{A}}^{\mathsf{Semi\text{-}Ad\text{-}Func\text{-}Sel\text{-}IND}}(1^\lambda, b)$

1. ***Parameters****:* $\mathcal{A}$ *takes as input* 1^λ*, and outputs a function size* $1^{\ell_{\mathcal{F}}}$*, an input size* $1^{\ell_{\mathcal{X}}}$*, and an output size* $1^{\ell_{\mathcal{Y}}}$*.*
2. ***Public Key****: Compute* $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}})$ *and send* mpk *to* $\mathcal{A}$*.*
3. ***Function Queries****: The following can be repeated any polynomial number of times:*

   (a) *$\mathcal{A}$ outputs a function query $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$*
   (b) $\mathsf{sk}_f \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}, f)$
   (c) *Send sk_f to $\mathcal{A}$*

4. ***Challenge Message:*** *$\mathcal{A}$ outputs a challenge message pair (x_0, x_1) where $x_0, x_1 \in \{0,1\}^{\ell_{\mathcal{X}}}$.*

5. ***Challenge Ciphertext:*** *Compute $\mathsf{ct} \leftarrow \mathsf{FE.Enc}(\mathsf{mpk}, x_b)$ and send ct to $\mathcal{A}$.*

6. ***Experiment Outcome:*** *$\mathcal{A}$ outputs a bit b'. The output of the experiment is set to 1 if $b = b'$ and $f(x_0) = f(x_1)$ for all functions f queried by the adversary.*

Remark 3.5. Adaptive-IND-security, which we do not formally define, is the same as semi-adaptive-function-selective-IND security except that we allow the adversary to make additional function queries after receiving the challenge ciphertext.

3.1.2 Secret-Key Functional Encryption

We can also define FE in the secret-key setting.

Definition 3.6 (Secret-Key Functional Encryption)**.** *Secret-key FE is the same as public-key FE except that* Setup *only outputs a master secret key and* Enc *requires the master secret key instead of the (non-existent) master public key. We formally define this in Appendix B.2.*

Remark 3.7. We can analogously define our public-key definitions of security in the secret-key setting. The only difference is that we do not give the (non-existent) master public key to the adversary and will therefore allow the adversary to submit multiple challenge message pairs. Note that semi-adaptive-function-selective-IND security is simply called function-selective-IND security in the secret-key setting. We formally define these security definitions in Appendix B.2.

In the secret-key setting, we can also achieve function privacy.

Definition 3.8 (Function-Private-Selective-IND-Security)**.** *A secret-key functional encryption scheme* FE *for* $\mathsf{P/Poly}$ *is function-private-selective-IND-secure if there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$ and every PPT adversary $\mathcal{A}$,*

$$\left|\Pr[\mathsf{SKExpt}_{\mathcal{A}}^{\mathsf{Func\text{-}Priv\text{-}Sel\text{-}IND}}(1^\lambda, 0) = 1] - \Pr[\mathsf{SKExpt}_{\mathcal{A}}^{\mathsf{Func\text{-}Priv\text{-}Sel\text{-}IND}}(1^\lambda, 1) = 1]\right| \le \mu(\lambda)$$

where for each $b \in \{0,1\}$ and $\lambda \in \mathbb{N}$, we define

$\mathsf{SKExpt}_{\mathcal{A}}^{\mathsf{Func\text{-}Priv\text{-}Sel\text{-}IND}}(1^\lambda, b)$

1. ***Parameters:*** *$\mathcal{A}$ takes as input 1^λ, and outputs a function size $1^{\ell_{\mathcal{F}}}$, an input size $1^{\ell_{\mathcal{X}}}$, and an output size $1^{\ell_{\mathcal{Y}}}$.*

2. ***Challenge Messages:*** *$\mathcal{A}$ outputs challenge message pairs $\{(x_{0,i}, x_{1,i})\}_{i\in[T]}$ for some T chosen by the adversary where $x_{0,i}, x_{1,i} \in \{0,1\}^{\ell_{\mathcal{X}}}$ for all $i \in [T]$.*

3. ***Setup and Challenge Ciphertexts:***

   (a) $\mathsf{msk} \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}})$
   (b) *For $i \in [T]$, compute $\mathsf{ct}_i \leftarrow \mathsf{FE.Enc}(\mathsf{msk}, x_{b,i})$ and send ct_i to $\mathcal{A}$.*

4. ***Function Queries:*** *The following can be repeated any polynomial number of times:*

   (a) *$\mathcal{A}$ outputs a function query pair (f_0, f_1) where $f_0, f_1 \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$*
   (b) $\mathsf{sk}_f \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}, f_b)$
   (c) *Send sk_f to $\mathcal{A}$*

5. ***Experiment Outcome:*** *$\mathcal{A}$ outputs a bit b'. The output of the experiment is set to 1 if $b = b'$ and $f_0(x_{0,i}) = f_1(x_{1,i})$ for all function pairs (f_0, f_1) queried by the adversary and all $i \in [T]$.*

3.1.3 Single-Key, Single-Ciphertext Security

Definition 3.9 (Single-Key, Single-Ciphertext Security)**.** *We can add the modifier "single-key. single-ciphertext" to any of our security definitions. This is a weakening of the security definition where we only require security against an adversary who is restricted to making only one function query and submitting only one challenge message pair in the relevant security game.*

3.1.4 Strong-Compactness

Additionally, we might also want our FE scheme to be *strongly-compact.*[8] Intuitively, this means that the sizes and running times of the setup and encryption algorithms are independent of the sizes of the circuits for which function keys are produced.

Definition 3.10 (Strong-Compactness)**.** *An FE scheme* $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.Enc}, \mathsf{FE.KeyGen}, \mathsf{FE.Dec})$ *for* $\mathsf{P/Poly}$ *is said to be* strongly-compact *if there exist PPT algorithms* $\mathsf{FE.Setup}^*, \mathsf{FE.Enc}^*$ *such that for all polynomials p, for all large enough $\lambda, \ell_{\mathcal{X}}$, we have that for all $\ell_{\mathcal{F}}, \ell_{\mathcal{Y}} \leq p(\lambda + \ell_{\mathcal{X}})$, the following holds:*

- $\mathsf{FE.Setup}(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}})$ *is identically distributed to* $\mathsf{FE.Setup}^*(1^\lambda, 1^{\ell_{\mathcal{X}}})$
- *For all* $\mathsf{mpk} \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}})$ *and all $x \in \{0,1\}^{\ell_{\mathcal{X}}}$,*
  $\mathsf{FE.Enc}(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}}, \mathsf{mpk}, x)$ *is identically distributed to* $\mathsf{FE.Enc}^*(1^\lambda, 1^{\ell_{\mathcal{X}}}, \mathsf{mpk}, x)$

We will often abuse notation and write $\mathsf{FE.Setup}$ *to mean* $\mathsf{FE.Setup}^*$ *and write* $\mathsf{FE.Enc}$ *to mean* $\mathsf{FE.Enc}^*$.

[8] We call it strong-compactness since the usual notion of compactness found in the literature only requires the encryption algorithm to not grow with the function size.

4 Streaming Functional Encryption

We now define our notion of streaming functional encryption which is an FE scheme for streaming functions. First, we define a streaming function.

Definition 4.1 (Streaming Function). *A streaming function with state space $\mathcal{S}$, input space $\mathcal{X}$, and output space $\mathcal{Y}$ is a function $f : \mathcal{X} \times \mathcal{S} \to \mathcal{Y} \times \mathcal{S}$.*

- *We define the **output** of f on $x = x_1 \dots x_n \in \mathcal{X}^n$ (denoted $f(x)$) to be $y = y_1 \dots y_n \in \mathcal{Y}^n$ where*[9] *we have $\mathsf{st}_1 = \perp$ and*

$$(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$$

Definition 4.2 (Streaming Function Class). *The streaming function class $\mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$ is the set of all streaming functions f that have a description $\widehat{f} \in \{0,1\}^{\ell_{\mathcal{F}}}$, state space $\mathcal{S} = \{0,1\}^{\ell_{\mathcal{S}}}$, input space $\mathcal{X} = \{0,1\}^{\ell_{\mathcal{X}}}$, and output space $\mathcal{Y} = \{0,1\}^{\ell_{\mathcal{Y}}}$.*

Definition 4.3 (Public-Key Streaming FE). *A public-key streaming functional encryption scheme for $\mathsf{P/Poly}$ is a tuple of PPT algorithms $\mathsf{sFE} = (\mathsf{Setup}, \mathsf{EncSetup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ defined as follows:*[10]

- $\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}})$*: takes as input the security parameter λ, a function size $\ell_{\mathcal{F}}$, a state size $\ell_{\mathcal{S}}$, an input size $\ell_{\mathcal{X}}$, and an output size $\ell_{\mathcal{Y}}$, and outputs the master public key* mpk *and the master secret key* msk.

- $\mathsf{EncSetup}(\mathsf{mpk})$*: takes as input the master public key* mpk *and outputs an encryption state* $\mathsf{Enc.st}$.

- $\mathsf{Enc}(\mathsf{mpk}, \mathsf{Enc.st}, i, x_i)$*: takes as input the master public key* mpk*, an encryption state* $\mathsf{Enc.st}$*, an index i, and a message $x_i \in \{0,1\}^{\ell_{\mathcal{X}}}$ and outputs an encryption* ct_i *of x_i.*

- $\mathsf{KeyGen}(\mathsf{msk}, f)$*: takes as input the master secret key* msk *and a function $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$ and outputs a function key* sk_f.

- $\mathsf{Dec}(\mathsf{sk}_f, \mathsf{Dec.st}_i, i, \mathsf{ct}_i)$*: where for each function key* sk_f*,* $\mathsf{Dec}(\mathsf{sk}_f, \cdot, \cdot, \cdot)$ *is a streaming function that takes as input a state* $\mathsf{Dec.st}_i$*, an index i, and an encryption* ct_i *and outputs a new state* $\mathsf{Dec.st}_{i+1}$ *and an output $y_i \in \{0,1\}^{\ell_{\mathcal{Y}}}$.*

sFE *satisfies **correctness** if for all polynomials p, there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$, all $\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}} \le p(\lambda)$, all $n \in [2^\lambda]$, all $x = x_1 \dots x_n$ where each $x_i \in \{0,1\}^{\ell_{\mathcal{X}}}$, and all $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$,*

$$\Pr\left[\overline{\mathsf{Dec}}(\mathsf{sk}_f, \mathsf{ct}_x) = f(x) : \begin{array}{c} (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}}), \\ \mathsf{ct}_x \leftarrow \overline{\mathsf{Enc}}(\mathsf{mpk}, x) \\ \mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f) \end{array}\right] \ge 1 - \mu(\lambda)$$

where we define[11]

- *the output of* $\overline{\mathsf{Enc}}(\mathsf{mpk}, x)$ *to be* $\mathsf{ct}_x = (\mathsf{ct}_i)_{i \in [n]}$ *produced by sampling* $\mathsf{Enc.st} \leftarrow \mathsf{EncSetup}(\mathsf{mpk})$ *and then computing $ct_i \leftarrow$* $\mathsf{Enc}(\mathsf{mpk}, \mathsf{Enc.st}, i, x_i)$ *for $i \in [n]$.*

- *the output of* $\overline{\mathsf{Dec}}(\mathsf{sk}_f, \mathsf{ct}_x)$ *to be $y = (y_i)_{i \in [n]}$ where* $(y_i, \mathsf{Dec.st}_{i+1}) = \mathsf{Dec}(\mathsf{sk}_f, \mathsf{Dec.st}_i, i, \mathsf{ct}_i)$

[9] We assume that unless specified otherwise, all streaming functions have the same starting state $\perp$ (or the all zero string) which is included in their state space.

[10] We also allow $\mathsf{Enc}, \mathsf{EncSetup}, \mathsf{KeyGen}$, and Dec to additionally receive parameters $1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}}$ as input, but omit them from our notation for convenience.

[11] As with all streaming functions, we assume that $\mathsf{Dec.st}_1 = \perp$ by default.

Efficiency. We require our streaming FE schemes to be *streaming efficient*, meaning that the runtime of our algorithms should not depend on the total length n of the message $x = x_1 \dots x_n$ that we wish to encrypt. More formally, we require that the size and runtime of all algorithms of sFE on security parameter λ, function size $\ell_{\mathcal{F}}$, state size $\ell_{\mathcal{S}}$, input size $\ell_{\mathcal{X}}$, and output size $\ell_{\mathcal{Y}}$ are $\mathsf{poly}(\lambda, \ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}})$.

Definition 4.4 (Secret-Key Streaming FE)**.** *Secret-key streaming FE is the same as public-key streaming FE except that* Setup *only outputs a master secret key and* $\mathsf{EncSetup}$ *and* Enc *require the master secret key instead of the (non-existent) master public key. We formally define this in Appendix C.*

Remark 4.5. We can also define a relaxed variant of streaming FE in which the encryption function is also a streaming function that takes as input the master public key, a state $\mathsf{Enc.st}_i$, an index i, and an input x_i, and outputs a new state $\mathsf{Enc.st}_{i+1}$, and an encryption ct_i of x_i. We define this notion in Appendix C.

4.1 Security

All of our definitions of security for streaming FE are exactly the same as the definitions of security for regular FE except that in the security games,

1. The adversary additionally outputs a state size parameter $1^{\ell_{\mathcal{S}}}$.
2. We allow function queries for streaming functions in $\mathcal{F}[\lambda, \ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$.
3. We allow the challenge message query pairs to be $(x^{(0)}, x^{(1)})$ where $x^{(0)} = x_1^{(0)} \dots x_n^{(0)}$ and $x^{(1)} = x_1^{(1)} \dots x_n^{(1)}$ for some length $n \in \mathbb{N}$ chosen by the adversary and where each $x_i^0, x_i^1 \in \{0,1\}^{\ell_{\mathcal{X}}}$.
4. We replace $\mathsf{Enc}(\mathsf{mpk}, x)$ with $\overline{\mathsf{Enc}}(\mathsf{mpk}, x)$ as defined in Definition 4.3.

As an example, we define the following:

Definition 4.6 (Semi-Adaptive-Function-Selective-IND-Security)**.** *A public-key streaming FE scheme* sFE *for* $\mathsf{P/Poly}$ *is semi-adaptive-function-selective-IND-secure if there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$ and all PPT adversaries $\mathcal{A}$,*

$$\left|\Pr[\mathsf{sFE\text{-}Expt}_{\mathcal{A}}^{\mathsf{Semi\text{-}Ad\text{-}Func\text{-}Sel\text{-}IND}}(1^\lambda, 0) = 1] - \Pr[\mathsf{sFE\text{-}Expt}_{\mathcal{A}}^{\mathsf{Semi\text{-}Ad\text{-}Func\text{-}Sel\text{-}IND}}(1^\lambda, 1) = 1]\right| \leq \mu(\lambda)$$

where for each $b \in \{0,1\}$ and $\lambda \in \mathbb{N}$, we define

$\mathsf{sFE\text{-}Expt}_{\mathcal{A}}^{\mathsf{Semi\text{-}Ad\text{-}Func\text{-}Sel\text{-}IND}}(1^\lambda, b)$

1. ***Parameters:*** *$\mathcal{A}$ takes as input 1^λ, and outputs a function size $1^{\ell_{\mathcal{F}}}$, a state size $1^{\ell_{\mathcal{S}}}$, an input size $1^{\ell_{\mathcal{X}}}$, and an output size $1^{\ell_{\mathcal{Y}}}$.*
2. ***Public Key:*** *Compute* $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{sFE.Setup}(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}})$ *and send* mpk *to $\mathcal{A}$.*
3. ***Function Queries:*** *The following can be repeated any polynomial number of times:*
   (a) *$\mathcal{A}$ outputs a streaming function query $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$*
   (b) $\mathsf{sk}_f \leftarrow \mathsf{sFE.KeyGen}(\mathsf{msk}, f)$

(c) *Send* sk_f *to* $\mathcal{A}$

4. ***Challenge Message:*** *$\mathcal{A}$ outputs a challenge message pair* $(x^{(0)}, x^{(1)})$ *where* $x^{(0)} = x_1^{(0)} \dots x_n^{(0)}$ *and* $x^{(1)} = x_1^{(1)} \dots x_n^{(1)}$ *for some length* $n \in \mathbb{N}$ *chosen by the adversary and where each* $x_i^{(0)}, x_i^{(1)} \in \{0,1\}^{\ell_{\mathcal{X}}}$.

5. ***Challenge Ciphertext:*** *Compute* $\mathsf{ct} \leftarrow \mathsf{sFE}.\overline{\mathsf{Enc}}(\mathsf{mpk}, x^{(b)})$ *and send* ct *to* $\mathcal{A}$.

6. ***Experiment Outcome:*** *$\mathcal{A}$ outputs a bit b'. The output of the experiment is set to* 1 *if* $b = b'$ *and* $f(x^{(0)}) = f(x^{(1)})$ *for all functions f queried by the adversary.*

Remark 4.7. Our definition of security above requires all elements of the challenge streams to be given before the adversary receives any ciphertexts. However, we can actually achieve a slightly more adaptive notion of security where the challenge stream messages can depend on the ciphertexts given for the previous stream values. In particular, in the security game above, the **Challenge Message** and **Challenge Ciphertext** phases can be replaced with one where the adversary iteratively outputs the next challenge message pair $(x_i^{(0)}, x_i^{(1)})$ and receives the next challenge ciphertext: an encryption of $x_i^{(b)}$. We can prove this stronger notion of security using the same proof already in the paper, with only minor modifications. In particular, we modify any intermediate definitions of security to also have this property and reformat the hybrids accordingly.

The rest of the security definitions in both the secret-key and public-key settings follow analogously.

We also define a weak notion of simulation security in the secret-key setting.

Definition 4.8 (Single-Key, Single-Ciphertext, Function-Selective-SIM-Security)**.** *A secret-key streaming FE scheme* sFE *for* $\mathsf{P/Poly}$ *is single-key, single-ciphertext, function-selective-SIM-secure if there exists a PPT simulator* Sim *and a negligible function μ such that for all $\lambda \in \mathbb{N}$ and all PPT adversaries $\mathcal{A}$,*

$$\left|\Pr[\mathsf{RealExpt}_{\mathcal{A}}^{\mathsf{One\text{-}Func\text{-}Sel\text{-}SIM}}(1^\lambda) = 1] - \Pr[\mathsf{IdealExpt}_{\mathcal{A},\mathsf{Sim}}^{\mathsf{One\text{-}Func\text{-}Sel\text{-}SIM}}(1^\lambda) = 1]\right| \leq \mu(\lambda)$$

where for $\lambda \in \mathbb{N}$, we define

$\mathsf{RealExpt}_{\mathcal{A}}^{\mathsf{One\text{-}Func\text{-}Sel\text{-}SIM}}(1^\lambda)$

1. ***Parameters:*** *$\mathcal{A}$ takes as input 1^λ, and outputs a function size $1^{\ell_{\mathcal{F}}}$, a state size $1^{\ell_{\mathcal{S}}}$, an input size $1^{\ell_{\mathcal{X}}}$, and an output size $1^{\ell_{\mathcal{Y}}}$.*

2. ***Setup:*** $\mathsf{msk} \leftarrow \mathsf{sFE.Setup}(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}})$

3. ***Function Query:***

   (a) *$\mathcal{A}$ outputs a streaming function query* $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$.

   (b) $\mathsf{sk}_f \leftarrow \mathsf{sFE.KeyGen}(\mathsf{msk}, f)$

   (c) *Send* sk_f *to* $\mathcal{A}$.

4. ***Message Query:***

   (a) *$\mathcal{A}$ outputs a message x where $x = x_1 \dots x_n$ for some $n \in \mathbb{N}$ chosen by the adversary and where each* $x_i \in \{0,1\}^{\ell_{\mathcal{X}}}$.

(b) $\mathsf{ct} \leftarrow \mathsf{sFE}.\overline{\mathsf{Enc}}(\mathsf{msk}, x)$

(c) *Send* ct *to* $\mathcal{A}$.

5. ***Experiment Outcome****:* $\mathcal{A}$ *outputs a bit* b *which is the output of the experiment.*

$\mathsf{IdealExpt}^{\mathsf{One\text{-}Func\text{-}Sel\text{-}SIM}}_{\mathcal{A},\mathsf{Sim}}(1^\lambda)$

1. ***Parameters****:* $\mathcal{A}$ *takes as input* 1^λ*, and outputs a function size* $1^{\ell_\mathcal{F}}$*, a state size* $1^{\ell_\mathcal{S}}$*, an input size* $1^{\ell_\mathcal{X}}$*, and an output size* $1^{\ell_\mathcal{Y}}$*.* Sim *receives* $(1^\lambda, 1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{S}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}})$.

2. ***Function Query****:*

   (a) $\mathcal{A}$ *outputs a streaming function query* $f \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$.

   (b) Sim *receives* f *and outputs a function key* sk_f.

   (c) *Send* sk_f *to* $\mathcal{A}$.

3. ***Message Query****:*

   (a) $\mathcal{A}$ *outputs a message* x *where* $x = x_1 \dots x_n$ *for some* $n \in \mathbb{N}$ *chosen by the adversary and where each* $x_i \in \{0,1\}^{\ell_\mathcal{X}}$.

   (b) Sim *receives* $(1^n, f(x))$ *and outputs a ciphertext* ct.

   (c) *Send* ct *to* $\mathcal{A}$.

4. ***Experiment Outcome****:* $\mathcal{A}$ *outputs a bit* b *which is the output of the experiment.*

Remark 4.9. In the secret-key setting, single-key, single-ciphertext, function-selective-SIM security implies single-key, single-ciphertext, function-selective-IND security.

5 Single-Key, Single-Ciphertext, SIM-secure, Secret-Key Streaming FE

In this section, we construct our main building block: a single-key, single-ciphertext, function-selective-SIM-secure, secret-key sFE scheme. We prove the following:

Theorem 5.1. *Assuming a strongly-compact, selective-IND-secure, secret-key FE scheme for* $\mathsf{P/Poly}$*, there exists a single-key, single-ciphertext, function-selective-SIM-secure, secret-key sFE scheme for* $\mathsf{P/Poly}$.

Please refer to the technical overview (Section 2) for a high level overview of our construction. To prove Theorem 5.1, we build an sFE scheme from the following tools, which as we show below, can each be instantiated using a strongly-compact, selective-IND-secure, secret-key FE scheme for $\mathsf{P/Poly}$.

Tools.

- $\mathsf{PRF} = (\mathsf{PRF.Setup}, \mathsf{PRF.Eval})$: A secure pseudorandom function family.
- $\mathsf{PRF2} = (\mathsf{PRF2.Setup}, \mathsf{PRF2.Eval})$: A secure pseudorandom function family.
- $\mathsf{Sym} = (\mathsf{Sym.Setup}, \mathsf{Sym.Enc}, \mathsf{Sym.Dec})$: A secure symmetric key encryption scheme.
- $\mathsf{Sym}' = (\mathsf{Sym}'.\mathsf{Setup}, \mathsf{Sym}'.\mathsf{Enc}, \mathsf{Sym}'.\mathsf{Dec})$: A secure symmetric key encryption scheme.
- $\mathsf{OneCompFE} = (\mathsf{OneCompFE.Setup}, \mathsf{OneCompFE.Enc}, \mathsf{OneCompFE.KeyGen}, \mathsf{OneCompFE.Dec})$: A *strongly-compact*, single-key, single-ciphertext, *selective*-IND-secure, secret-key FE scheme for $\mathsf{P/Poly}$.
- $\mathsf{OneFSFE} = (\mathsf{OneFSFE.Setup}, \mathsf{OneFSFE.Enc}, \mathsf{OneFSFE.KeyGen}, \mathsf{OneFSFE.Dec})$: A single-key, single-ciphertext, *function-selective*-IND-secure, secret-key FE scheme for $\mathsf{P/Poly}$.

Instantiation of the Tools. Let SKFE be a strongly-compact, selective-IND-secure, secret-key FE scheme for $\mathsf{P/Poly}$.

- We can build $\mathsf{PRF}, \mathsf{PRF2}, \mathsf{Sym}, \mathsf{Sym}'$ from any one-way-function using standard cryptographic techniques (e.g. [Gol01, Gol09]). As FE implies one-way-functions, then we can build these from SKFE.
- SKFE already satisfies the compactness and security requirements needed for $\mathsf{OneCompFE}$.
- We can first build a function-private-selective-IND-secure, secret-key FE scheme FPFE for $\mathsf{P/Poly}$ by using the function-privacy transformation of [BS18] on SKFE. As observed in [BS18], a single-key, single-ciphertext, function-private-selective-IND-secure, secret-key FE scheme for $\mathsf{P/Poly}$ is also a (non-compact) single-key, single-ciphertext, function-selective-IND-secure, secret-key FE scheme for $\mathsf{P/Poly}$ as we can simply exchange the roles of the functions and messages using universal circuits. Thus, FPFE can be used to build $\mathsf{OneFSFE}$.

5.1 Parameters

On security parameter λ, function size $\ell_{\mathcal{F}}$, state size $\ell_{\mathcal{S}}$, input size $\ell_{\mathcal{X}}$, and output size $\ell_{\mathcal{Y}}$, we will instantiate our primitives with the following parameters:

- PRF: We instantiate PRF with input size $\lambda+2$ and output size λ. This means that we will use the following setup algorithm: $\mathsf{PRF.Setup}(1^\lambda, 1^{\lambda+2}, 1^\lambda)$.
- $\mathsf{PRF2}$: We instantiate $\mathsf{PRF2}$ with input size λ and output size $\ell_{\mathcal{S}}$. This means that we will use the following setup algorithm: $\mathsf{PRF2.Setup}(1^\lambda, 1^\lambda, 1^{\ell_{\mathcal{S}}})$.
- Sym: We instantiate Sym with message size $\ell_{\mathsf{Sym}.m_\lambda}$ for $\ell_{\mathsf{Sym}.m_\lambda}$ defined below. This means that we will use the following setup algorithm: $\mathsf{Sym.Setup}(1^\lambda, 1^{\ell_{\mathsf{Sym}.m_\lambda}})$.
- Sym': We instantiate Sym' with message size $\ell_{\mathsf{Sym}'.m_\lambda}$ for $\ell_{\mathsf{Sym}'.m_\lambda}$ defined below. This means that we will use the following setup algorithm: $\mathsf{Sym}'.\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{Sym}'.m_\lambda}})$
- $\mathsf{OneCompFE}$: We instantiate $\mathsf{OneCompFE}$ with function size ℓ_{h_λ}, input size $\ell_{m'_\lambda}$, and output size $\ell_{\mathsf{Sym}'.m_\lambda}$ for parameters $\ell_{h_\lambda}, \ell_{m'_\lambda}, \ell_{\mathsf{Sym}'.m_\lambda}$ defined below. This means that we will use the following algorithms:
  - $\mathsf{OneCompFE.Setup}(1^\lambda, 1^{\ell_{m'_\lambda}})$
  - $\mathsf{OneCompFE.Enc}(1^\lambda, 1^{\ell_{m'_\lambda}}, \cdot, \cdot)$
  - $\mathsf{OneCompFE.KeyGen}(1^\lambda, 1^{\ell_{h_\lambda}}, 1^{\ell_{m'_\lambda}}, 1^{\ell_{\mathsf{Sym}'.m_\lambda}}, \cdot, \cdot)$
  - $\mathsf{OneCompFE.Dec}(1^\lambda, 1^{\ell_{h_\lambda}}, 1^{\ell_{m'_\lambda}}, 1^{\ell_{\mathsf{Sym}'.m_\lambda}}, \cdot, \cdot)$

  Observe that $\mathsf{OneCompFE.Setup}$ and $\mathsf{OneCompFE.Enc}$ do not require the function size or output size as input since $\mathsf{OneCompFE}$ is strongly-compact.
- $\mathsf{OneFSFE}$: We instantiate $\mathsf{OneFSFE}$ with function size ℓ_{g_λ}, input size ℓ_{m_λ}, and output size $\ell_{\mathsf{Sym}.m_\lambda}$ for parameters $\ell_{g_\lambda}, \ell_{m_\lambda}, \ell_{\mathsf{Sym}.m_\lambda}$ defined below. This means that we will use the following algorithms:
  - $\mathsf{OneFSFE.Setup}(1^\lambda, 1^{\ell_{g_\lambda}}, 1^{\ell_{m_\lambda}}, 1^{\ell_{\mathsf{Sym}.m_\lambda}})$
  - $\mathsf{OneFSFE.Enc}(1^\lambda, 1^{\ell_{g_\lambda}}, 1^{\ell_{m_\lambda}}, 1^{\ell_{\mathsf{Sym}.m_\lambda}}, \cdot, \cdot)$
  - $\mathsf{OneFSFE.KeyGen}(1^\lambda, 1^{\ell_{g_\lambda}}, 1^{\ell_{m_\lambda}}, 1^{\ell_{\mathsf{Sym}.m_\lambda}}, \cdot, \cdot)$
  - $\mathsf{OneFSFE.Dec}(1^\lambda, 1^{\ell_{g_\lambda}}, 1^{\ell_{m_\lambda}}, 1^{\ell_{\mathsf{Sym}.m_\lambda}}, \cdot, \cdot)$

Notation. For notational convenience, when the parameters are understood, we will often omit the security, input size, output size, message size, function size, or state size parameters from each of the algorithms listed above.

Remark 5.2. We assume without loss of generality that for security parameter λ, all algorithms only require randomness of length λ. If the original algorithm required additional randomness, we can replace it with a new algorithm that first expands the λ bits of randomness using a PRG of appropriate stretch and then runs the original algorithm. Note that this replacement does not affect the security of the above schemes (as long as $\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}$ are polynomial in λ) and preserves the strong-compactness of $\mathsf{OneCompFE}$.

Parameter Table. We now show how to define our parameters without circular dependencies. Each parameter in the table below may depend on any of the parameters above it. The table is continued on the next page.

Table 1: Parameters

Size	**Description**	**Variables of that Size**
λ	The security parameter and the size of all randomness used.	$r_{\mathsf{msk}_i}, r'_{\mathsf{msk}_i}, r_{k_i}, r'_{k_i}, r'_{\mathsf{Enc}_i}, r_{\mathsf{KeyGen}_i}$
$\ell_{\mathcal{F}}$	The size of functions in $\mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$.	f
$\ell_{\mathcal{X}}$	The size of inputs to functions in $\mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$.	x_i
$\ell_{\mathcal{Y}}$	The size of outputs of functions in $\mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$.	y_i, θ_i, ψ_i
$\ell_{\mathcal{S}}$	The size of states of functions in $\mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$.	$p_i, \mathsf{st}_i, \widetilde{\mathsf{st}}_i$
$\ell_{m'_\lambda} = \ell_{\mathcal{F}} + \ell_{\mathcal{S}} + 3\lambda + 1$	The size of input messages for $\mathsf{OneCompFE}$.	$(f, \widetilde{\mathsf{st}}_i, r_{\mathsf{msk}_i}, r_{\mathsf{KeyGen}_i}, \alpha'_i, r'_{k_i})$
$\ell_{\mathsf{Setup}'_\lambda} = \mathsf{poly}(\lambda, \ell_{m'_\lambda})$	The size of the setup algorithm for $\mathsf{OneCompFE}$.[12]	$\mathsf{OneCompFE.Setup}$
$\ell_{\mathsf{Enc}'_\lambda} = \mathsf{poly}(\lambda, \ell_{m'_\lambda})$	The size of the encryption algorithm for $\mathsf{OneCompFE}$.	$\mathsf{OneCompFE.Enc}$
$\ell_{\mathsf{ct}'_\lambda} = \mathsf{poly}(\lambda, \ell_{m'_\lambda})$	The size of ciphertexts for $\mathsf{OneCompFE}$.	ct'_i

[12] Since $\mathsf{OneCompFE}$ is strongly-compact, $\ell_{\mathsf{Setup}'_\lambda}$, $\ell_{\mathsf{Enc}'_\lambda}$ and $\ell_{\mathsf{ct}'_\lambda}$ can be defined based only on the security parameter λ and input message length $\ell_{m'_\lambda}$, without regard for the function length ℓ_{h_λ} and output length $\ell_{\mathsf{Sym}'.m_\lambda}$ which will be defined later.

Table 2: Parameters (continued)

Size	**Description**	**Variables of that Size**
$\ell_{\mathsf{Sym}.m_\lambda} = \ell_{\mathcal{Y}} + \ell_{\mathsf{ct}'_\lambda}$	The size of input messages to Sym and the size of outputs of functions for $\mathsf{OneFSFE}$.	$(\theta_i, \mathsf{ct}'_{i+1}), (y_i, \mathsf{ct}'_{i+1})$ $(\theta_i \oplus \psi_i, \mathsf{ct}'_{i+1})$
$\ell_{\mathsf{Sym.Setup}_\lambda} = \mathsf{poly}(\lambda, \ell_{\mathsf{Sym}.m_\lambda})$	The size of the setup algorithm of Sym.	$\mathsf{Sym.Setup}$
$\ell_{\mathsf{Sym.Dec}_\lambda} = \mathsf{poly}(\lambda, \ell_{\mathsf{Sym}.m_\lambda})$	The size of the decryption algorithm of Sym.	$\mathsf{Sym.Dec}$
$\ell_{\mathsf{Sym.ct}_\lambda} = \mathsf{poly}(\lambda, \ell_{\mathsf{Sym}.m_\lambda})$	The size of ciphertexts of Sym.	c_i
$\ell_{m_\lambda} = \ell_{\mathcal{X}} + 2\ell_{\mathcal{S}} + 5\lambda + \ell_{\mathcal{Y}} + 1$	The size of input messages for $\mathsf{OneFSFE}$.	$(x_i, p_i, p_{i+1}, r'_{\mathsf{msk}_{i+1}}, r'_{\mathsf{Enc}_{i+1}}, r_{\mathsf{msk}_{i+1}}, r_{\mathsf{KeyGen}_{i+1}}, \alpha_i, r_{k_i}, \psi_i)$
$\ell_{g_\lambda} = \mathsf{poly}(\lambda, \ell_{m_\lambda}, \ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}, \ell_{\mathsf{Sym.ct}_\lambda}, \ell_{\mathsf{Setup}'_\lambda}, \ell_{\mathsf{Enc}'_\lambda}, \ell_{\mathsf{Sym.Setup}_\lambda}, \ell_{\mathsf{Sym.Dec}_\lambda})$	The size of functions for $\mathsf{OneFSFE}$. This is set to be the maximum size of $g_{f,\widetilde{\mathsf{st}}_i,c_i}$ defined in Figure 6 for any $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$, $\widetilde{\mathsf{st}}_i \in \{0,1\}^{\ell_{\mathcal{S}}}$, and c_i of size $\ell_{\mathsf{Sym.ct}_\lambda}$.	$g_{f,\widetilde{\mathsf{st}}_i,c_i}$
$\ell_{\mathsf{Setup}_\lambda} = \mathsf{poly}(\lambda, \ell_{m_\lambda}, \ell_{g_\lambda}, \ell_{\mathsf{Sym}.m_\lambda})$	The size of the setup algorithm for $\mathsf{OneFSFE}$.	$\mathsf{OneFSFE.Setup}$
$\ell_{\mathsf{KeyGen}_\lambda} = \mathsf{poly}(\lambda, \ell_{m_\lambda}, \ell_{g_\lambda}, \ell_{\mathsf{Sym}.m_\lambda})$	The size of the keygen algorithm for $\mathsf{OneFSFE}$.	$\mathsf{OneFSFE.KeyGen}$
$\ell_{\mathsf{sk}_\lambda} = \mathsf{poly}(\lambda, \ell_{m_\lambda}, \ell_{g_\lambda}, \ell_{\mathsf{Sym}.m_\lambda})$	The size of function keys for $\mathsf{OneFSFE}$.	sk_{g_i}
$\ell_{\mathsf{Sym}'.m_\lambda} = \ell_{\mathsf{sk}_\lambda}$	The size of input messages for Sym', and the size of outputs of functions for $\mathsf{OneCompFE}$.	sk_{g_i}
$\ell_{\mathsf{Sym}'.\mathsf{Setup}_\lambda} = \mathsf{poly}(\lambda, \ell_{\mathsf{Sym}'.m_\lambda})$	The size of the setup algorithm of Sym'.	$\mathsf{Sym}'.\mathsf{Setup}$
$\ell_{\mathsf{Sym}'.\mathsf{Dec}_\lambda} = \mathsf{poly}(\lambda, \ell_{\mathsf{Sym}'.m_\lambda})$	The size of the decryption algorithm of Sym'.	$\mathsf{Sym}'.\mathsf{Dec}$
$\ell_{\mathsf{Sym}'.\mathsf{ct}_\lambda} = \mathsf{poly}(\lambda, \ell_{\mathsf{Sym}'.m_\lambda})$	The size of ciphertexts of Sym'.	c'_i
$\ell_{h_\lambda} = \mathsf{poly}(\lambda, \ell_{m'_\lambda}, \ell_{\mathsf{Setup}_\lambda} \ell_{\mathsf{KeyGen}_\lambda}, \ell_{\mathsf{Sym}'.\mathsf{Setup}_\lambda}, \ell_{\mathsf{Sym}'.\mathsf{Dec}_\lambda} \ell_{\mathsf{Sym.ct}_\lambda}, \ell_{\mathsf{Sym.ct}'_\lambda})$	The size of functions for $\mathsf{OneCompFE}$. This is set to be the maximum size of h_{c_i,c'_i} defined in Figure 7 for any c_i of size $\ell_{\mathsf{Sym.ct}_\lambda}$ and c'_i of size $\ell_{\mathsf{Sym}'.\mathsf{ct}_\lambda}$.	h_{c_i,c'_i}

5.2 Construction

We now construct our streaming FE scheme One-sFE. Recall that for notational convenience, we may omit the security, input size, output size, message size, function size, or state size parameters from our setup and FE algorithms. For information on these parameters, please see the paramter section above.

- $\mathsf{One\text{-}sFE.Setup}(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}})$:

  1. $\mathsf{PRF}.K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$, $\mathsf{PRF2}.K \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$

  * Throughout, for $i \in [2^\lambda]$, we will define

  $$\begin{aligned} r_{\mathsf{msk}_i} &= \mathsf{PRF.Eval}(\mathsf{PRF}.K, (i, 0)) \\ \mathsf{msk}_i &= \mathsf{OneFSFE.Setup}(1^\lambda; r_{\mathsf{msk}_i}) \\ r'_{\mathsf{msk}_i} &= \mathsf{PRF.Eval}(\mathsf{PRF}.K, (i, 1)) \\ \mathsf{msk}'_i &= \mathsf{OneCompFE.Setup}(1^\lambda; r'_{\mathsf{msk}_i}) \\ r_{k_i} &= \mathsf{PRF.Eval}(\mathsf{PRF}.K, (i, 2)) \\ k_i &= \mathsf{Sym.Setup}(1^\lambda; r_{k_i}) \\ r'_{k_i} &= \mathsf{PRF.Eval}(\mathsf{PRF}.K, (i, 3)) \\ k'_i &= \mathsf{Sym}'.\mathsf{Setup}(1^\lambda; r'_{k_i}) \end{aligned}$$

  Observe that these can all be computed from $\mathsf{PRF}.K$ and i. We will also define

  $$p_i = \mathsf{PRF2.Eval}(\mathsf{PRF2}.K, i)$$

  which can be computed from $\mathsf{PRF2}.K$ and i.

  2. Output $\mathsf{MSK} = (\mathsf{PRF}.K, \mathsf{PRF2}.K)$

- $\mathsf{One\text{-}sFE.EncSetup}(\mathsf{MSK})$: Output $\mathsf{Enc.st} = \perp$.

- $\mathsf{One\text{-}sFE.Enc}(\mathsf{MSK}, \mathsf{Enc.st}, i, x_i)$

  1. Parse $\mathsf{MSK} = (\mathsf{PRF}.K, \mathsf{PRF2}.K)$.
  2. Compute $\mathsf{msk}_i, p_i, p_{i+1}, r'_{\mathsf{msk}_{i+1}}, r_{\mathsf{msk}_{i+1}}, k_i, k'_i, \mathsf{msk}'_i$ from $\mathsf{PRF}.K, \mathsf{PRF2}.K$.
  3. $r'_{\mathsf{Enc}_{i+1}}, r_{\mathsf{KeyGen}_{i+1}} \leftarrow \{0,1\}^\lambda$
  4. $\mathsf{ct}_i \leftarrow \mathsf{OneFSFE.Enc}(\mathsf{msk}_i, (x_i, p_i, p_{i+1}, r'_{\mathsf{msk}_{i+1}}, r'_{\mathsf{Enc}_{i+1}}, r_{\mathsf{msk}_{i+1}}, r_{\mathsf{KeyGen}_{i+1}}, 0, 0^\lambda, 0^{\ell_{\mathcal{Y}}}))$
  5. If $i = 1$, output $\mathsf{CT}_1 = \mathsf{ct}_1$.
  6. If $i > 1$
     (a) $c_i \leftarrow \mathsf{Sym.Enc}(k_i, 0^{\ell_{\mathsf{Sym}.m_\lambda}})$
     (b) $c'_i \leftarrow \mathsf{Sym}'.\mathsf{Enc}(k'_i, 0^{\ell_{\mathsf{Sym}'.m_\lambda}})$
     (c) Let $h_i = h_{c_i, c'_i}$ as defined in Figure 7.
     (d) $\mathsf{sk}'_{h_i} \leftarrow \mathsf{OneCompFE.KeyGen}(\mathsf{msk}'_i, h_i)$
     (e) Output $\mathsf{CT}_i = (\mathsf{ct}_i, \mathsf{sk}'_{h_i})$

- $\mathsf{One\text{-}sFE.KeyGen}(\mathsf{MSK}, f)$

  1. Parse $\mathsf{MSK} = (\mathsf{PRF}.K, \mathsf{PRF2}.K)$.

2. Compute msk_1, k_1, p_1 from $\mathsf{PRF}.K, \mathsf{PRF2}.K$.
3. $c_1 \leftarrow \mathsf{Sym.Enc}(k_1, 0^{\ell_{\mathsf{Sym}.m_\lambda}})$
4. $\widetilde{\mathsf{st}}_1 = p_1$ (Here, we assume $\mathsf{st}_1 = 0^{\ell_S}$ for all streaming functions so that $\mathsf{st}_1 = \widetilde{\mathsf{st}}_1 \oplus p_1$.)
5. Let $g_1 = g_{f,\widetilde{\mathsf{st}}_1,c_1}$ as defined in Figure 6.
6. $\mathsf{sk}_{g_1} \leftarrow \mathsf{OneFSFE.KeyGen}(\mathsf{msk}_1, g_1)$
7. Output $\mathsf{SK}_f = \mathsf{sk}_{g_1}$.

```
$g_{f,\widetilde{\mathsf{st}}_i,c_i}(x_i, p_i, p_{i+1}, r'_{\mathsf{msk}_{i+1}}, r'_{\mathsf{Enc}_{i+1}}, r_{\mathsf{msk}_{i+1}}, r_{\mathsf{KeyGen}_{i+1}}, \alpha_i, r_{k_i}, \psi_i)$:
  • If $\alpha_i = 0$,
      1. $\mathsf{st}_i = \widetilde{\mathsf{st}}_i \oplus p_i$
      2. $(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$
      3. $\widetilde{\mathsf{st}}_{i+1} = \mathsf{st}_{i+1} \oplus p_{i+1}$
      4. $\mathsf{msk}'_{i+1} = \mathsf{OneCompFE.Setup}(1^\lambda; r'_{\mathsf{msk}_{i+1}})$
      5. $\mathsf{ct}'_{i+1} = \mathsf{OneCompFE.Enc}(\mathsf{msk}'_{i+1}, (f, \widetilde{\mathsf{st}}_{i+1}, r_{\mathsf{msk}_{i+1}}, r_{\mathsf{KeyGen}_{i+1}}, 0, 0^\lambda); r'_{\mathsf{Enc}_{i+1}})$
      6. Output $(y_i, ct'_{i+1})$.
  • Else
      1. $k_i = \mathsf{Sym.Setup}(1^\lambda; r_{k_i})$
      2. $(\theta_i, \mathsf{ct}'_{i+1}) = \mathsf{Sym.Dec}(k_i, c_i)$.
      3. Output $(\theta_i \oplus \psi_i, \mathsf{ct}'_{i+1})$.
```

Figure 6

```
$h_{c_i,c'_i}(f, \widetilde{\mathsf{st}}_i, r_{\mathsf{msk}_i}, r_{\mathsf{KeyGen}_i}, \alpha'_i, r'_{k_i})$
  • If $\alpha'_i = 0$,
      1. $\mathsf{msk}_i = \mathsf{OneFSFE.Setup}(1^\lambda; r_{\mathsf{msk}_i})$
      2. Let $g_i = g_{f,\widetilde{\mathsf{st}}_i,c_i}$ as defined in Figure 6.
      3. $\mathsf{sk}_{g_i} = \mathsf{OneFSFE.KeyGen}(\mathsf{msk}_i, g_i; r_{\mathsf{KeyGen}_i})$
      4. Output $sk_{g_i}$.
  • Else
      1. $k'_i = \mathsf{Sym'.Setup}(1^\lambda; r'_{k_i})$
      2. Output $\mathsf{sk}_{g_i} = \mathsf{Sym'.Dec}(k'_i, c'_i)$.
```

Figure 7

- $\mathsf{One\text{-}sFE.Dec}(\mathsf{SK}_f, \mathsf{Dec.ST}_i, i, \mathsf{CT}_i)$:
  1. If $i = 1$
     (a) Parse $\mathsf{SK}_1 = \mathsf{sk}_{g_1}$ and $\mathsf{CT}_1 = \mathsf{ct}_1$
     (b) $(y_1, \mathsf{ct}'_2) = \mathsf{OneFSFE.Dec}(\mathsf{sk}_{g_1}, \mathsf{ct}_1)$
     (c) Output $(y_1, \mathsf{Dec.ST}_2 = \mathsf{ct}'_2)$
  2. If $i > 1$
     (a) Parse $\mathsf{Dec.ST}_i = \mathsf{ct}'_i$ and $\mathsf{CT}_i = (\mathsf{ct}_i, \mathsf{sk}'_{h_i})$.
     (b) $\mathsf{sk}_{g_i} = \mathsf{OneCompFE.Dec}(\mathsf{sk}'_{h_i}, \mathsf{ct}'_i)$.
     (c) $(y_i, \mathsf{ct}'_{i+1}) = \mathsf{OneFSFE.Dec}(\mathsf{sk}_{g_i}, \mathsf{ct}_i)$
     (d) Output $(y_i, \mathsf{Dec.ST}_{i+1} = \mathsf{ct}'_{i+1})$

5.3 Correctness and Efficiency

Efficiency. Using our discussion above on parameters, it is easy to see that the size and runtime of all algorithms of $\mathsf{One\text{-}sFE}$ on security parameter λ, function size $\ell_{\mathcal{F}}$, state size $\ell_{\mathcal{S}}$, input size $\ell_{\mathcal{X}}$, and output size $\ell_{\mathcal{Y}}$ are $\mathsf{poly}(\lambda, \ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}})$.

Correctness Intuition. Each sk_{g_i} and ct_i can be combined via $\mathsf{OneFSFE}$ decryption to obtain y_i and ct'_{i+1}. We obtain $\{\mathsf{ct}_i\}_{i\in[n]}$ from the ciphertext for x, and get the first function key sk_{g_1} as the function key for f. For $i > 1$, we can use $\mathsf{OneCompFE}$ decryption to iteratively combine the ct'_i generated by the previous step with the sk'_{h_i} given in the ciphertext to get the next sk_{g_i}. This lets us continue the process for all $i \in [n]$ and recover $y = y_1 \dots y_n$.

Correctness. More formally, let p be any polynomial and consider any λ and any $\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}} \leq p(\lambda)$. Let SK_f be a function key for function $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$, and let $\{\mathsf{CT}_i\}_{i\in[n]}$ be a ciphertext for x where $x = x_1 \dots x_n$ for some $n \in [2^\lambda]$ and where each $x_i \in \{0,1\}^{\ell_{\mathcal{X}}}$. When $i = 1$, by correctness of $\mathsf{OneFSFE}$, except with negligible probability,

$$\begin{aligned}
\mathsf{One\text{-}sFE.Dec}(\mathsf{SK}_f, \mathsf{Enc.ST}_1, \mathsf{CT}_1) &= \mathsf{One\text{-}sFE.Dec}(\mathsf{sk}_{g_1}, \bot, \mathsf{ct}_1)\\
&= \mathsf{OneFSFE.Dec}(\mathsf{sk}_{g_1}, \mathsf{ct}_1)\\
&= g_{f,\mathsf{st}_1\oplus p_1, c_1}(x_1, p_1, p_2, r'_{\mathsf{msk}_2}, r'_{\mathsf{Enc}_2}, r_{\mathsf{msk}_2}, r_{\mathsf{KeyGen}_2}, 0, 0^\lambda, 0^{\ell_{\mathcal{Y}}})\\
&= (y_1, \mathsf{Dec.ST}_2 = \mathsf{ct}'_2)
\end{aligned}$$

where $(y_1, \mathsf{st}_2) = f(x_1, \mathsf{st}_1)$ and $\mathsf{ct}'_2 = \mathsf{OneCompFE.Enc}(\mathsf{msk}'_2, (f, \mathsf{st}_2 \oplus p_2, r_{\mathsf{msk}_2}, r_{\mathsf{KeyGen}_2}, 0, 0^\lambda); r'_{\mathsf{Enc}_2})$
When $i = 2$, by correctness of $\mathsf{OneCompFE}$ and $\mathsf{OneFSFE}$, except with negligible probablity,

$$\begin{aligned}
\mathsf{One\text{-}sFE.Dec}(\mathsf{SK}_f, \mathsf{Dec.ST}_2, \mathsf{CT}_2) &= \mathsf{One\text{-}sFE.Dec}(\mathsf{sk}_{g_1}, \mathsf{ct}'_2, (\mathsf{ct}_2, \mathsf{sk}'_{h_2}))\\
&= \mathsf{OneFSFE.Dec}(\mathsf{OneCompFE.Dec}(\mathsf{sk}'_{h_2}, \mathsf{ct}'_2), \mathsf{ct}_2)\\
&= \mathsf{OneFSFE.Dec}(h_{c_2, c'_2}(f, \mathsf{st}_2 \oplus p_2, r_{\mathsf{msk}_2}, r_{\mathsf{KeyGen}_2}, 0, 0^\lambda), \mathsf{ct}_2)\\
&= \mathsf{OneFSFE.Dec}(\mathsf{OneFSFE.KeyGen}(\mathsf{msk}_2, g_{f,\mathsf{st}_2\oplus p_2, c_2}; r_{\mathsf{KeyGen}_2}), \mathsf{ct}_2)\\
&= \mathsf{OneFSFE.Dec}(\mathsf{sk}_{g_2}, \mathsf{ct}_2)\\
&= g_{f,\mathsf{st}_2\oplus p_2, c_2}(x_2, p_2, p_3, r'_{\mathsf{msk}_3}, r'_{\mathsf{Enc}_3}, r_{\mathsf{msk}_3}, r_{\mathsf{KeyGen}_3}, 0, 0^\lambda, 0^{\ell_{\mathcal{Y}}})\\
&= (y_2, \mathsf{Dec.ST}_3 = \mathsf{ct}'_3)
\end{aligned}$$

where $(y_2, \mathsf{st}_3) = f(x_2, \mathsf{st}_2)$ and $\mathsf{ct}'_3 = \mathsf{OneCompFE.Enc}(\mathsf{msk}'_3, (f, \mathsf{st}_3 \oplus p_3, r_{\mathsf{msk}_3}, r_{\mathsf{KeyGen}_3}, 0, 0^\lambda); r'_{\mathsf{Enc}_3})$.
Similarly, by induction, for $i > 2$, except with negligible probability,

$$\mathsf{One\text{-}sFE.Dec}(\mathsf{SK}_f, \mathsf{Dec.ST}_i, \mathsf{CT}_i) = (y_i, \mathsf{Dec.ST}_{i+1} = \mathsf{ct}'_{i+1})$$

where $(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$ and $\mathsf{ct}'_{i+1} = \mathsf{OneCompFE.Enc}(\mathsf{msk}'_{i+1}, (f, \mathsf{st}_{i+1} \oplus p_{i+1}, r_{\mathsf{msk}_{i+1}}, r_{\mathsf{KeyGen}_{i+1}}, 0, 0^\lambda); r'_{\mathsf{Enc}_{i+1}})$.
Thus, we correctly output $y = y_1 \ldots y_n$ where $(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$ and $\mathsf{st}_1 = 0^{\ell_S}$.

5.4 Security

We use a hybrid argument to prove that our scheme is single-key, single-ciphertext, function-selective-SIM-secure (see Definition 4.8). Our PPT simulator is defined in the final hybrid ($\mathbf{Hybrid}_8^{\mathcal{A}}$) of the formal security proof. We need to prove that the real world security game $\mathbf{Hybrid}_1^{\mathcal{A}}$ is indistinguishable from the ideal world security game $\mathbf{Hybrid}_8^{\mathcal{A}}$.

5.4.1 Proof Overview

To build intuition, we provide a brief overview of each hybrid below.

- $\mathbf{Hybrid}_1^{\mathcal{A}}$: This is the real world experiment. The adversary first receives the security parameter and chooses the function size, state size, input size, and output size. Then, the adversary submits a function query and receives a function key. Next, the adversary submits a message query and receives the corresponding ciphertext. Finally, the adversary outputs a single bit which is the outcome of this experiment.

- $\mathbf{Hybrid}_2^{\mathcal{A}}$: We exchange the PRF randomness for true randomness. Instead of generating the randomness for $\mathsf{msk}_i, \mathsf{msk}'_i, k_i, k'_i, p_i$ using the master secret key, which consists of PRF keys $\mathsf{PRF}.K$ and $\mathsf{PRF2}.K$, we generate these values using true randomness. The indistinguishability of $\mathbf{Hybrid}_1^{\mathcal{A}}$ and $\mathbf{Hybrid}_2^{\mathcal{A}}$ holds by the security of PRF and $\mathsf{PRF2}$.

- $\mathbf{Hybrid}_3^{\mathcal{A}}$: We adjust the way we sample p_i and $\widetilde{\mathsf{st}}_i$ so that each $\widetilde{\mathsf{st}}_i$ is now sampled uniformly at random. For each i, to hide the intermediate state st_i, our previous hybrid padded st_i with a one time pad p_i to get $\widetilde{\mathsf{st}}_i = \mathsf{st}_i \oplus p_i$. The value $\widetilde{\mathsf{st}}_i$ can then be leaked (and is in fact leaked) as long as p_i remains hidden. In this hybrid, rather than computing $\widetilde{\mathsf{st}}_i = \mathsf{st}_i \oplus p_i$ for a random pad p_i, we compute $p_i = \mathsf{st}_i \oplus \widetilde{\mathsf{st}}_i$ for a random value $\widetilde{\mathsf{st}}_i$. This lets us use $\widetilde{\mathsf{st}}_i$ before knowing the value of the true state st_i. It is easy to see that $\mathbf{Hybrid}_2^{\mathcal{A}}$ and $\mathbf{Hybrid}_3^{\mathcal{A}}$ are identically distributed.

- $\mathbf{Hybrid}_4^{\mathcal{A}}$: We hardcode in values for the $\alpha_i = 1$ branch of g_i. For each i, we hardcode into c_i the values $(y_i, \mathsf{ct}'_{i+1})$ that are output by $g_i = g_{f,\widetilde{\mathsf{st}},c_i}$ on the $\alpha_i = 0$ branch if we run it on the input generated by the challenge message x. (i.e. $c_i \leftarrow \mathsf{Sym}.\mathsf{Enc}(\mathsf{Sym}.k_i, (y_i, \mathsf{ct}'_{i+1}))$). The objective is to later use the security of $\mathsf{OneFSFE}$ to switch to the $\alpha_i = 1$ branch of g_i, which does not require knowledge of x in the input. We also need to ensure that this hardcoding can be done before knowing the value of x (or $y = f(x)$) as we must output $\mathsf{SK}_f = \mathsf{sk}_{g_1}$ before learning x. Observe that the output value of g_i in the $\alpha_i = 0$ branch depends only on y_i, f, $\widetilde{\mathsf{st}}_{i+1}$, and randomness. f is known at this stage and the randomness can be pre-computed. Additionally, because of our previous hybrid, we can compute $\widetilde{\mathsf{st}}_{i+1}$ before knowing x. To deal with y_i, instead of encrypting y_i directly, we encrypt a random value θ_i. We can then correct this value later by substituting in an appropriate $\psi_i = \theta_i \oplus y_i$ into the ciphertext when we switch to the $\alpha_i = 1$ branch. The indistinguishability of $\mathbf{Hybrid}_3^{\mathcal{A}}$ and $\mathbf{Hybrid}_4^{\mathcal{A}}$ holds by the security of Sym.

- $\mathbf{Hybrid}_5^{\mathcal{A}}$: We hardcode in values for the $\alpha'_i = 1$ branch of h_i. For each i, we hardcode into c'_i the value sk_{g_i} that would be output by $h_i = h_{c_i,c'_i}$ in the $\alpha'_i = 0$ branch if we were to run it on the input generated by the challenge message x. (i.e. $c'_i \leftarrow \mathsf{Sym}'.\mathsf{Enc}(\mathsf{Sym}'.k_i, \mathsf{sk}_{g_i})$). The objective is to later use the security of $\mathsf{OneCompFE}$ to switch to the $\alpha'_i = 1$ branch of h_i, which does not require knowledge of x in the input. We also need to ensure that this hardcoding can be done without knowing the value of x so that we can later achieve simulation security.

Observe that the output value of h_i in the $\alpha'_i = 0$ branch depends only on $g_i = g_{f,\widetilde{\mathsf{st}}_i,c_i}$ and randomness. As we showed in our previous hybrid, we can compute each g_i without knowing x and can precompute the randomness, so there is no dependence on x in this hardcoding of c'_i. The indistinguishability of $\mathbf{Hybrid}_4^{\mathcal{A}}$ and $\mathbf{Hybrid}_5^{\mathcal{A}}$ holds by the security of Sym'.

- We will now go through the following hybrids for $k = 1$ to q where $q = q(\lambda)$ is the runtime of $\mathcal{A}$ so that $q(\lambda) \geq n$ for any challenge message $x = x_1 \dots x_n$ output by $\mathcal{A}$ on security parameter λ.

  - $\mathbf{Hybrid}_{6,k,1}^{\mathcal{A}}(1^\lambda)$: For the k^{th} ciphertext ct_k, instead of generating

$$\mathsf{ct}_k \leftarrow \mathsf{OneFSFE.Enc}(\mathsf{msk}_k, (x_k, p_k, p_{k+1}, r'_{\mathsf{msk}_{k+1}}, r'_{\mathsf{Enc}_{k+1}}, r_{\mathsf{msk}_{k+1}}, r_{\mathsf{KeyGen}_{k+1}}, 0, 0^\lambda, 0^{\ell_{\mathcal{Y}}})),$$

    we generate

$$\mathsf{ct}_k \leftarrow \mathsf{OneFSFE.Enc}(\mathsf{msk}_k, (0^{\ell_{\mathcal{X}}}, 0^{\ell_{\mathcal{S}}}, 0^{\ell_{\mathcal{S}}}, 0^\lambda, 0^\lambda, 0^\lambda, 0^\lambda, 1, r_{k_k}, \psi_k))$$

    where $\psi_k = \theta_k \oplus y_k$. Observe that the only function key generated under msk_k is

$$\mathsf{sk}_{g_k} \leftarrow \mathsf{OneFSFE.KeyGen}(\mathsf{msk}_k, g_{f,\widetilde{\mathsf{st}}_k,c_k}; r_{\mathsf{KeyGen}_k})$$

    Additionally, because we have hardcoded the correct output value into c_k in a previous hybrid,

$$\begin{aligned} &g_{f,\widetilde{\mathsf{st}}_k,c_k}(x_k, p_k, p_{i+1}, r'_{\mathsf{msk}_{k+1}}, r'_{\mathsf{Enc}_{k+1}}, r_{\mathsf{msk}_{k+1}}, r_{\mathsf{KeyGen}_{k+1}}, 0, 0^\lambda, 0^{\ell_{\mathcal{Y}}}) \\ &= g_{f,\widetilde{\mathsf{st}}_k,c_k}(0^{\ell_{\mathcal{X}}}, 0^{\ell_{\mathcal{S}}}, 0^{\ell_{\mathcal{S}}}, 0^\lambda, 0^\lambda, 0^\lambda, 0^\lambda, 1, r_{k_k}, \psi_k) \end{aligned}$$

    Thus we should be able to swap these ciphertexts by the security of $\mathsf{OneFSFE}$ as long as msk_k, r_{msk_k}, and r_{KeyGen_k} remain hidden. Now, except for their appearances in ct_k and sk_{g_k}, msk_k appears nowhere else in the hybrid and r_{msk_k} and r_{KeyGen_k} only appear in ct_{k-1} and in ct'_k (which is used to hardcode c'_{k-1}). (For $k = 1$, r_{msk_1} and r_{KeyGen_1} appear nowhere else as there is no ct_0, and ct'_1 is not used.) However, since we are going through these hybrids iteratively from $k = 1$ to q, then we will do $\mathbf{Hybrid}_{6,k-1,1}^{\mathcal{A}}$ before this hybrid which means that ct_{k-1} will no longer contain r_{msk_k} and r_{KeyGen_k}. Additionally, we will also do $\mathbf{Hybrid}_{6,k-1,2}^{\mathcal{A}}$ before this hybrid which will remove r_{msk_k} and r_{KeyGen_k} from ct'_k (as will be seen shortly). Thus, we have removed all other occurrences of msk_k, r_{msk_k}, r_{KeyGen_k} except for ct_k and sk_{g_k}, so we can argue indistinguishability by the security of $\mathsf{OneFSFE}$.

  - $\mathbf{Hybrid}_{6,k,2}^{\mathcal{A}}(1^\lambda)$: For the $(k+1)^{th}$ ciphertext ct'_{k+1} (which is used to hardcode c'_k), instead of generating

$$\mathsf{ct}'_{k+1} \leftarrow \mathsf{OneCompFE.Enc}(\mathsf{msk}'_{k+1}, (f, \widetilde{\mathsf{st}}_{k+1}, r_{\mathsf{msk}_{k+1}}, r_{\mathsf{KeyGen}_{k+1}}, 0, 0^\lambda); r'_{\mathsf{Enc}_{k+1}}),$$

    we will generate

$$\mathsf{ct}'_{k+1} \leftarrow \mathsf{OneCompFE.Enc}(\mathsf{msk}'_{k+1}, (0^{\ell_{\mathcal{F}}}, 0^{\ell_{\mathcal{S}}}, 0^\lambda, 0^\lambda, 1, r'_{k_k}); r'_{\mathsf{Enc}_{k+1}})$$

    Observe that the only function key generated under msk'_{k+1} is

$$\mathsf{sk}'_{h_{k+1}} \leftarrow \mathsf{OneCompFE.KeyGen}(\mathsf{msk}'_{k+1}, h_{c_{k+1},c'_{k+1}})$$

Additionally, because we have hardcoded the correct output value into c'_{k+1} in a previous hybrid

$$\begin{aligned} & h_{c_{k+1},c'_{k+1}}(f, \widetilde{\mathsf{st}}_{k+1}, r_{\mathsf{msk}_{k+1}}, r_{\mathsf{KeyGen}_{k+1}}, 0, 0^\lambda) \\ & = h_{c_{k+1},c'_{k+1}}(0^{\ell_\mathcal{F}}, 0^{\ell_\mathcal{S}}, 0^\lambda, 0^\lambda, 1, r'_{k_k}) \end{aligned}$$

Thus we should be able to swap these ciphertexts by the security of $\mathsf{OneCompFE}$ as long as msk'_{k+1}, $r'_{\mathsf{msk}_{k+1}}$, and $r'_{\mathsf{Enc}_{k+1}}$ remain hidden. Now, except for their appearances in ct'_{k+1} and $\mathsf{sk}'_{h_{k+1}}$, msk'_{k+1} appears nowhere else in the hybrid and $r'_{\mathsf{msk}_{k+1}}$ and $r'_{\mathsf{Enc}_{k+1}}$ only appear in ct_k. However, since we are going through these hybrids iteratively from $k = 1$ to q, then we will do $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,1}$ before this hybrid which means that ct_k will no longer contain $r'_{\mathsf{msk}_{k+1}}$ and $r'_{\mathsf{Enc}_{k+1}}$. Thus, we have removed all other occurrences of msk'_{k+1}, $r'_{\mathsf{msk}_{k+1}}$, $r_{\mathsf{Enc}'_{k+1}}$ except for ct'_{k+1} and $\mathsf{sk}'_{h_{k+1}}$, so we can argue indistinguishability using the security of $\mathsf{OneFSFE}$.

- $\mathbf{Hybrid}^{\mathcal{A}}_7$: This is the same as $\mathbf{Hybrid}^{\mathcal{A}}_{6,q,2}$ where $q = q(\lambda)$ is the runtime of $\mathcal{A}$. We write it explicitly to make the simulator in the next hybrid easier to understand.
- $\mathbf{Hybrid}^{\mathcal{A}}_8$: We formally write the previous hybrid as a simulator. This hybrid acts identically to the previous one. Observe that as $q(\lambda) \geq n$ for all $x = x_1 \dots x_n$ output by $\mathcal{A}$ on security parameter λ, then we will use the $\alpha_i = \alpha'_i = 1$ branches of all g_i and h_i. Thus, to generate our ciphertexts ct_i and ct'_i, we only need to know $y = f(x)$ (as ψ_i depends on y_i) and don't need to know x. Furthermore to generate our function keys sk_{g_i} and sk'_{h_i}, we only need the programmed values of c_i and c'_i, which also do not depend on x. Thus we can simulate this hybrid with y instead of x.

5.4.2 Formal Proof

We now formally prove security via a hybrid argument. The first hybrid, $\mathbf{Hybrid}^{\mathcal{A}}_1$, is the real world game. The last hybrid, $\mathbf{Hybrid}^{\mathcal{A}}_8$, is the ideal world game with our simulator. We prove that these hybrids are computationally indistinguishable. We defer the definition of our simulator to the final hybrid.

Remark 5.3. In steps 3 (**Compute Randomness**) and 4 (**Compute sk_{g_i} and sk'_{h_i}**) of all of the hybrids in this proof, we refer to the length n of the challenge message x before it is revealed to the challenger or simulator by the adversary in step 6 (**Challenge Message**). This is technically incorrect. We write it this way, however, as it greatly increases proof readability.

We can easily correct our hybrids by doing the following: Before receiving the challenge message x, the challenger or simulator will only run step 3 (**Compute Randomness**) up to $i = 2$ and step 4 (**Compute sk_{g_i} and sk'_{h_i}**) up to $i = 1$. This suffices for computing the function key. After the challenger or simulator receives the challenge message x and learns the value of n, then it can finish the remainder of these steps.

This issue is not relevant in our reductions, as our reductions will compute these steps for $i \in [q]$ where $q = q(\lambda)$ is the running time of $\mathcal{A}$ and thus $q(\lambda) \geq n$ for any $x = x_1 \dots x_n$ output by $\mathcal{A}$ on security parameter λ.

Hybrid$_1^{\mathcal{A}}(1^\lambda)$: This is the real world experiment. Though we have reordered some steps for the sake of the proof, this does not affect the output of the experiment.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs a function size $1^{\ell_{\mathcal{F}}}$, a state size $1^{\ell_{\mathcal{S}}}$, an input size $1^{\ell_{\mathcal{X}}}$, and an output size $1^{\ell_{\mathcal{Y}}}$.

2. **Function Query:** $\mathcal{A}$ outputs a streaming function query $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$.

3. **Compute Randomness:**
   (a) $\mathsf{PRF}.K \leftarrow \mathsf{PRF.Setup}(1^\lambda), \mathsf{PRF2}.K \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.
   (b) For $i \in [n+1]$,
      i. Compute $r_{\mathsf{msk}_i}, r'_{\mathsf{msk}_i}, r_{k_i}, r_{k'_i}, p_i$ from $\mathsf{PRF}.K, \mathsf{PRF2}.K$.
      ii. $\mathsf{msk}_i \leftarrow \mathsf{OneFSFE.Setup}(1^\lambda; r_{\mathsf{msk}_i})$, $\mathsf{msk}'_i \leftarrow \mathsf{OneCompFE.Setup}(1^\lambda; r'_{\mathsf{msk}_i})$,
      $k_i \leftarrow \mathsf{Sym.Setup}(1^\lambda; r_{k_i})$, $k'_i \leftarrow \mathsf{Sym'.Setup}(1^\lambda; r'_{k_i})$
      iii. $r'_{\mathsf{Enc}_i}, r_{\mathsf{KeyGen}_i} \leftarrow \{0,1\}^\lambda$

4. **Compute sk_{g_i} and sk'_{h_i}:**
   (a) For $i \in [n]$,
      i. $c_i \leftarrow \mathsf{Sym.Enc}(k_i, 0^{\ell_{\mathsf{Sym}.m_\lambda}})$
      ii. If $i = 1$
         A. $\widetilde{\mathsf{st}}_1 = p_1$
         B. $\mathsf{sk}_{g_1} \leftarrow \mathsf{OneFSFE.KeyGen}(\mathsf{msk}_1, g_{f,\widetilde{\mathsf{st}}_1,c_1})$ for $g_{f,\widetilde{\mathsf{st}}_1,c_1}$ as defined in Figure 6.
      iii. If $i > 1$
         A. $c'_i \leftarrow \mathsf{Sym.Enc}(k'_i, 0^{\ell_{\mathsf{Sym'}.m_\lambda}})$
         B. $\mathsf{sk}'_{h_i} \leftarrow \mathsf{OneCompFE.KeyGen}(\mathsf{msk}'_i, h_{c_i,c'_i})$ for h_{c_i,c'_i} as defined in Figure 7.

5. **Function Key:** Send $\mathsf{SK}_f = \mathsf{sk}_{g_1}$ to the adversary.

6. **Challenge Message:** $\mathcal{A}$ outputs a challenge message $x = x_1 \dots x_n$ for some $n \in \mathbb{N}$ and where each $x_i \in \{0,1\}^{\ell_{\mathcal{X}}}$.

7. **Challenge Ciphertext:**
   (a) For $i \in [n]$,
      i. $\mathsf{ct}_i \leftarrow \mathsf{OneFSFE.Enc}(\mathsf{msk}_i, (x_i, p_i, p_{i+1}, r'_{\mathsf{msk}_{i+1}}, r'_{\mathsf{Enc}_{i+1}}, r_{\mathsf{msk}_{i+1}}, r_{\mathsf{KeyGen}_{i+1}}, 0, 0^\lambda, 0^{\ell_{\mathcal{Y}}}))$
      ii. If $i = 1$, $\mathsf{CT}_1 = \mathsf{ct}_1$. Else, $\mathsf{CT}_i = (\mathsf{ct}_i, \mathsf{sk}'_{h_i})$
   (b) Send $\mathsf{CT} = (\mathsf{CT}_i)_{i \in [n]}$ to the adversary.

8. **Adversary's Output:** $\mathcal{A}$ outputs a bit b which is the outcome of the experiment.

Hybrid$_2^{\mathcal{A}}(1^\lambda)$: We exchange the randomness generated by $\mathsf{PRF}.K$ and $\mathsf{PRF2}.K$ with true randomness.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs a function size $1^{\ell_{\mathcal{F}}}$, a state size $1^{\ell_{\mathcal{S}}}$, an input size $1^{\ell_{\mathcal{X}}}$, and an output size $1^{\ell_{\mathcal{Y}}}$.

2. **Function Query:** $\mathcal{A}$ outputs a streaming function query $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$.

3. **Compute Randomness:**

   (a) ~~$\mathsf{PRF}.K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$, $\mathsf{PRF2}.K \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.~~

   (b) For $i \in [n+1]$,

      i. $r_{\mathsf{msk}_i}, r'_{\mathsf{msk}_i}, r_{k_i}, r'_{k_i}, r'_{\mathsf{Enc}_i}, r_{\mathsf{KeyGen}_i} \leftarrow \{0,1\}^\lambda$

      ii. $\mathsf{msk}_i \leftarrow \mathsf{OneFSFE.Setup}(1^\lambda; r_{\mathsf{msk}_i})$, $\mathsf{msk}'_i \leftarrow \mathsf{OneCompFE.Setup}(1^\lambda; r'_{\mathsf{msk}_i})$, $k_i \leftarrow \mathsf{Sym.Setup}(1^\lambda; r_{k_i})$, $k'_i \leftarrow \mathsf{Sym'.Setup}(1^\lambda; r'_{k_i})$

      iii. $p_i \leftarrow \{0,1\}^{\ell_{\mathcal{S}}}$

4. **Compute** sk_{g_i} **and** sk'_{h_i}:

   (a) For $i \in [n]$,

      i. $c_i \leftarrow \mathsf{Sym.Enc}(k_i, 0^{\ell_{\mathsf{Sym}.m_\lambda}})$

      ii. If $i = 1$

         A. $\widetilde{\mathsf{st}}_1 = p_1$

         B. $\mathsf{sk}_{g_1} \leftarrow \mathsf{OneFSFE.KeyGen}(\mathsf{msk}_1, g_{f,\widetilde{\mathsf{st}}_1,c_1})$ for $g_{f,\widetilde{\mathsf{st}}_1,c_1}$ as defined in Figure 6.

      iii. If $i > 1$

         A. $c'_i \leftarrow \mathsf{Sym.Enc}(k'_i, 0^{\ell_{\mathsf{Sym'}.m_\lambda}})$

         B. $\mathsf{sk}'_{h_i} \leftarrow \mathsf{OneCompFE.KeyGen}(\mathsf{msk}'_i, h_{c_i,c'_i})$ for h_{c_i,c'_i} as defined in Figure 7.

5. **Function Key:** Send $\mathsf{SK}_f = \mathsf{sk}_{g_1}$ to the adversary.

6. **Challenge Message:** $\mathcal{A}$ outputs a challenge message $x = x_1 \ldots x_n$ for some $n \in \mathbb{N}$ and where each $x_i \in \{0,1\}^{\ell_{\mathcal{X}}}$.

7. **Challenge Ciphertext:**

   (a) For $i \in [n]$,

      i. $\mathsf{ct}_i \leftarrow \mathsf{OneFSFE.Enc}(\mathsf{msk}_i, (x_i, p_i, p_{i+1}, r'_{\mathsf{msk}_{i+1}}, r'_{\mathsf{Enc}_{i+1}}, r_{\mathsf{msk}_{i+1}}, r_{\mathsf{KeyGen}_{i+1}}, 0, 0^\lambda, 0^{\ell_{\mathcal{Y}}}))$

      ii. If $i = 1$, $\mathsf{CT}_1 = \mathsf{ct}_1$. Else, $\mathsf{CT}_i = (\mathsf{ct}_i, \mathsf{sk}'_{h_i})$

   (b) Send $\mathsf{CT} = (\mathsf{CT}_i)_{i \in [n]}$ to the adversary.

8. **Adversary's Output:** $\mathcal{A}$ outputs a bit b which is the outcome of the experiment.

Lemma 5.4. *If* PRF *and* $\mathsf{PRF2}$ *are secure PRFs, then for all PPT adversaries* $\mathcal{A}$*,*

$$\left| \Pr[\mathbf{Hybrid}_1^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_2^{\mathcal{A}}(1^\lambda) = 1] \right| \leq \mathsf{negl}(\lambda)$$

Proof. Let $\mathbf{Hybrid}_{1,2}^{\mathcal{A}}$ be the same as $\mathbf{Hybrid}_1^{\mathcal{A}}$ except that $\{r_{\mathsf{msk}_i}, r'_{\mathsf{msk}_i}, r_{k_i}, r'_{k_i}\}_{i\in[n+1]}$ are sampled uniformly at random, instead of using $\mathsf{PRF}.K$. Suppose for contradiction that there exists a PPT adversary $\mathcal{A}$ such that

$$\left|\Pr[\mathbf{Hybrid}_1^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_2^{\mathcal{A}}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda)$$

Then, either

$$\left|\Pr[\mathbf{Hybrid}_1^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_{1,2}^{\mathcal{A}}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \quad (1)$$

or

$$\left|\Pr[\mathbf{Hybrid}_{1,2}^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_2^{\mathcal{A}}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \quad (2)$$

Let $q = q(\lambda)$ be the running time of $\mathcal{A}$. Observe that $q = \mathsf{poly}(\lambda)$ as $\mathcal{A}$ is polytime and that $q(\lambda) \geq n$ for any challenge message $x = x_1 \dots x_n$ output by $\mathcal{A}$ on security parameter λ.
In case (1), we build a PPT adversary $\mathcal{B}$ that breaks the security of PRF. $\mathcal{B}$ first runs $\mathcal{A}$ on input 1^λ and receives parameters $1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{S}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}}$ and a function query $f \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$. $\mathcal{B}$ then sends input length parameter $1^{\lambda+2}$ and output length parameter 1^λ to its PRF challenger. $\mathcal{B}$ is then given oracle access to either $\mathsf{PRF.Eval}(\mathsf{PRF}.K, \cdot)$ for some $\mathsf{PRF}.K \leftarrow \mathsf{PRF.Setup}(1^\lambda, 1^{\lambda+2}, 1^\lambda)$ or to a uniformly random function $R \leftarrow \mathcal{R}_{\lambda+2,\lambda}$ where $\mathcal{R}_{\lambda+2,\lambda}$ is the set of all functions from $\{0,1\}^{\lambda+2}$ to $\{0,1\}^\lambda$. Then $\mathcal{B}$ computes $\{r_{\mathsf{msk}_i}, r'_{\mathsf{msk}_i}, r_{k_i}, r'_{k_i}\}_{i\in[q+1]}$ using its oracle on $\{(i,0),(i,1),(i,2),(i,3)\}_{i\in[q+1]}$ respectively and computes $\{\mathsf{msk}_i, \mathsf{msk}'_i, k_i, k'_i\}_{i\in[q+1]}$ from these values as in $\mathbf{Hybrid}_1^{\mathcal{A}}$. $\mathcal{B}$ samples $\mathsf{PRF2}.K \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$ and computes $\{p_i\}_{i\in[q+1]}$ from $\mathsf{PRF2}.K$ as in $\mathbf{Hybrid}_1^{\mathcal{A}}$. $\mathcal{B}$ samples $\{r'_{\mathsf{Enc}_i}, r_{\mathsf{KeyGen}_i}\}_{i\in[q+1]}$ uniformly at random. $\mathcal{B}$ computes sk_{g_1} and $\{\mathsf{sk}'_{h_i}\}_{i\in[q]\setminus\{1\}}$ as in $\mathbf{Hybrid}_1^{\mathcal{A}}$, sends $\mathsf{SK}_f = \mathsf{sk}_{g_1}$ to $\mathcal{A}$, and receives a challenge message x. $\mathcal{B}$ then computes the challenge ciphertext CT as in $\mathbf{Hybrid}_1^{\mathcal{A}}$, sends it to $\mathcal{A}$, and outputs whatever $\mathcal{A}$ outputs. Observe that if $\mathcal{B}$'s oracle was $\mathsf{PRF.Eval}(\mathsf{PRF}.K, \cdot)$, then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}_1^{\mathcal{A}}$, and if $\mathcal{B}$'s oracle was a uniform random function R, then $\mathcal{B}$ emulates $\mathbf{Hybrid}_{1,2}^{\mathcal{A}}$. Additionally, $\mathcal{B}$ does not need to use $\mathsf{PRF}.K$ to run this experiment. Thus, by Equation 1 this means that we break the security of PRF since

$$\begin{aligned}&\left|\Pr[\mathsf{Expt}_{\mathcal{B}}^{\mathsf{PRF}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Expt}_{\mathcal{B}}^{\mathsf{PRF}}(1^\lambda, 1) = 1]\right| \\ &= \left|\Pr[\mathbf{Hybrid}_1^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_{1,2}^{\mathcal{A}}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda)\end{aligned}$$

Similarly, in case (2), we can build a PPT adversary $\mathcal{B}2$ that breaks the security of $\mathsf{PRF2}$. $\mathcal{B}2$ first runs $\mathcal{A}$ on input 1^λ and receives parameters $1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{S}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}}$ and a function query $f \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$. $\mathcal{B}2$ then sends input length parameter 1^λ and output length parameter $1^{\ell_\mathcal{S}}$ to its $\mathsf{PRF2}$ challenger. $\mathcal{B}2$ is then given oracle access to either $\mathsf{PRF2.Eval}(\mathsf{PRF2}.K, \cdot)$ for some $\mathsf{PRF2}.K \leftarrow \mathsf{PRF2.Setup}(1^\lambda, 1^\lambda, 1^{\ell_\mathcal{S}})$ or to a uniformly random function $R2 \leftarrow \mathcal{R}2_{\lambda,\ell_\mathcal{S}}$ where $\mathcal{R}2_{\lambda,\ell_\mathcal{S}}$ is the set of all functions from $\{0,1\}^\lambda$ to $\{0,1\}^{\ell_\mathcal{S}}$. Then $\mathcal{B}2$ samples $\{r_{\mathsf{msk}_i}, r'_{\mathsf{msk}_i}, r_{k_i}, r'_{k_i}, r'_{\mathsf{Enc}_i}, r_{\mathsf{KeyGen}_i}\}_{i\in[q+1]}$ uniformly at random and computes $\{\mathsf{msk}_i, \mathsf{msk}'_i, k_i, k'_i\}_{i\in[q+1]}$ from these values as in $\mathbf{Hybrid}_1^{\mathcal{A}}$. For $i \in [q]$, $\mathcal{B}2$ sets p_i to be the value of its oracle on input i. $\mathcal{B}2$ computes sk_{g_1} and $\{\mathsf{sk}'_{h_i}\}_{i\in[q]\setminus\{1\}}$ as in $\mathbf{Hybrid}_1^{\mathcal{A}}$, sends $\mathsf{SK}_f = \mathsf{sk}_{g_1}$ to $\mathcal{A}$, and receives a challenge message x. $\mathcal{B}2$ then computes the challenge ciphertext CT as in $\mathbf{Hybrid}_1^{\mathcal{A}}$, sends it to $\mathcal{A}$, and outputs whatever $\mathcal{A}$ outputs. Observe that if $\mathcal{B}2$'s oracle was $\mathsf{PRF2.Eval}(\mathsf{PRF2}.K, \cdot)$, then $\mathcal{B}2$ exactly emulates $\mathbf{Hybrid}_{1,2}^{\mathcal{A}}$, and if $\mathcal{B}2$'s oracle was a uniform random function $R2$, then $\mathcal{B}2$ emulates $\mathbf{Hybrid}_2^{\mathcal{A}}$. Additionally, $\mathcal{B}$ does not need to use $\mathsf{PRF2}.K$ to run this experiment. Thus, by Equation 2 this means that we break the security of $\mathsf{PRF2}$ since

$$\begin{aligned}&\left|\Pr[\mathsf{Expt}_{\mathcal{B}}^{\mathsf{PRF2}}(1^\lambda,0)=1]-\Pr[\mathsf{Expt}_{\mathcal{B}}^{\mathsf{PRF2}}(1^\lambda,1)=1]\right|\\&=\left|\Pr[\mathbf{Hybrid}_{1,2}^{\mathcal{A}}(1^\lambda)=1]-\Pr[\mathbf{Hybrid}_{2}^{\mathcal{A}}(1^\lambda)=1]\right|>\mathsf{negl}(\lambda)\end{aligned}$$

□

Hybrid$_3^{\mathcal{A}}(1^\lambda)$: For each i, we now determine p_i by XOR-ing the true state st_i with a random value $\widetilde{\mathsf{st}}_i$.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs a function size $1^{\ell_{\mathcal{F}}}$, a state size $1^{\ell_{\mathcal{S}}}$, an input size $1^{\ell_{\mathcal{X}}}$, and an output size $1^{\ell_{\mathcal{Y}}}$.
2. **Function Query:** $\mathcal{A}$ outputs a streaming function query $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$.
3. **Compute Randomness:**
   (a) For $i \in [n+1]$,
      i. $r_{\mathsf{msk}_i}, r'_{\mathsf{msk}_i}, r_{k_i}, r'_{k_i}, r'_{\mathsf{Enc}_i}, r_{\mathsf{KeyGen}_i} \leftarrow \{0,1\}^\lambda$
      ii. $\mathsf{msk}_i \leftarrow \mathsf{OneFSFE.Setup}(1^\lambda; r_{\mathsf{msk}_i})$, $\mathsf{msk}'_i \leftarrow \mathsf{OneCompFE.Setup}(1^\lambda; r'_{\mathsf{msk}_i})$, $k_i \leftarrow \mathsf{Sym.Setup}(1^\lambda; r_{k_i})$, $k'_i \leftarrow \mathsf{Sym'.Setup}(1^\lambda; r'_{k_i})$
      iii. $\widetilde{\mathsf{st}}_i \leftarrow \{0,1\}^{\ell_{\mathcal{S}}}$
4. **Compute** sk_{g_i} **and** sk'_{h_i}:
   (a) For $i \in [n]$,
      i. $c_i \leftarrow \mathsf{Sym.Enc}(k_i, 0^{\ell_{\mathsf{Sym}.m_\lambda}})$
      ii. If $i = 1$
         A. ~~$\widetilde{\mathsf{st}}_1 = p_1$~~
         B. $\mathsf{sk}_{g_1} \leftarrow \mathsf{OneFSFE.KeyGen}(\mathsf{msk}_1, g_{f,\widetilde{\mathsf{st}}_1,c_1})$ for $g_{f,\widetilde{\mathsf{st}}_1,c_1}$ as defined in Figure 6.
      iii. If $i > 1$
         A. $c'_i \leftarrow \mathsf{Sym.Enc}(k'_i, 0^{\ell_{\mathsf{Sym'}.m_\lambda}})$
         B. $\mathsf{sk}'_{h_i} \leftarrow \mathsf{OneCompFE.KeyGen}(\mathsf{msk}'_i, h_{c_i,c'_i})$ for h_{c_i,c'_i} as defined in Figure 7.
5. **Function Key:** Send $\mathsf{SK}_f = \mathsf{sk}_{g_1}$ to the adversary.
6. **Challenge Message:** $\mathcal{A}$ outputs a challenge message $x = x_1 \ldots x_n$ for some $n \in \mathbb{N}$ and where each $x_i \in \{0,1\}^{\ell_{\mathcal{X}}}$.
7. **Compute** p_i**:**
   (a) $\mathsf{st}_1 = 0^{\ell_{\mathcal{S}}}$
   (b) For $i \in [n]$,
      i. $p_i = \widetilde{\mathsf{st}}_i \oplus \mathsf{st}_i$
      ii. $(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$
   (c) $p_{i+1} = \widetilde{\mathsf{st}}_{i+1} \oplus \mathsf{st}_{i+1}$
8. **Challenge Ciphertext:**
   (a) For $i \in [n]$,
      i. $\mathsf{ct}_i \leftarrow \mathsf{OneFSFE.Enc}(\mathsf{msk}_i, (x_i, p_i, p_{i+1}, r'_{\mathsf{msk}_{i+1}}, r'_{\mathsf{Enc}_{i+1}}, r_{\mathsf{msk}_{i+1}}, r_{\mathsf{KeyGen}_{i+1}}, 0, 0^\lambda, 0^{\ell_{\mathcal{Y}}}))$
      ii. If $i = 1$, $\mathsf{CT}_1 = \mathsf{ct}_1$. Else, $\mathsf{CT}_i = (\mathsf{ct}_i, \mathsf{sk}'_{h_i})$
   (b) Send $\mathsf{CT} = (\mathsf{CT}_i)_{i \in [n]}$ to the adversary.
9. **Adversary's Output:** $\mathcal{A}$ outputs a bit b which is the outcome of the experiment.

Lemma 5.5. *For all adversaries $\mathcal{A}$,*

$$\left|\Pr[\mathbf{Hybrid}_2^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_3^{\mathcal{A}}(1^\lambda) = 1]\right| = 0$$

Proof. The two hybrids are identically distributed. Observe that if $\widetilde{\mathsf{st}}_i$ is uniformly random and $p_i = \widetilde{\mathsf{st}}_i \oplus \mathsf{st}_i$, then p_i is also uniformly random. As $\widetilde{\mathsf{st}}_i$ is not used in these hybrids except to compute p_i (and except for $\widetilde{\mathsf{st}}_1 = p_1 \leftarrow \{0,1\}^{\ell_{\mathcal{S}}}$ which is the same in both hybrids), then the hybrids are identically distributed. $\square$

Hybrid$_4^{\mathcal{A}}(1^\lambda)$: For each i, we hardcode into c_i the values $(y_i, \mathsf{ct}'_{i+1})$ that are output by $g_i = g_{f,\widetilde{\mathsf{st}},c_i}$ on the $\alpha_i = 0$ branch if we run it on the input generated by the challenge message x. This will allow us to later switch to the $\alpha_i = 1$ branch in $g_i = g_{f,\widetilde{\mathsf{st}}_i,c_i}$ using the security of $\mathsf{OneFSFE}$. Observe that the values being hardcoded into c_i can be determined before knowing x.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs a function size $1^{\ell_{\mathcal{F}}}$, a state size $1^{\ell_{\mathcal{S}}}$, an input size $1^{\ell_{\mathcal{X}}}$, and an output size $1^{\ell_{\mathcal{Y}}}$.

2. **Function Query:** $\mathcal{A}$ outputs a streaming function query $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$.

3. **Compute Randomness:**

   (a) For $i \in [n+1]$,

   i. $r_{\mathsf{msk}_i}, r'_{\mathsf{msk}_i}, r_{k_i}, r'_{k_i}, r'_{\mathsf{Enc}_i}, r_{\mathsf{KeyGen}_i} \leftarrow \{0,1\}^\lambda$

   ii. $\mathsf{msk}_i \leftarrow \mathsf{OneFSFE.Setup}(1^\lambda; r_{\mathsf{msk}_i})$, $\mathsf{msk}'_i \leftarrow \mathsf{OneCompFE.Setup}(1^\lambda; r'_{\mathsf{msk}_i})$, $k_i \leftarrow \mathsf{Sym.Setup}(1^\lambda; r_{k_i})$, $k'_i \leftarrow \mathsf{Sym'.Setup}(1^\lambda; r'_{k_i})$

   iii. $\widetilde{\mathsf{st}}_i \leftarrow \{0,1\}^{\ell_{\mathcal{S}}}$

4. **Compute sk_{g_i} and sk'_{h_i}:**

   (a) For $i \in [n]$,

   i. $\theta_i \leftarrow \{0,1\}^{\ell_{\mathcal{Y}}}$

   ii. $\mathsf{ct}'_{i+1} \leftarrow \mathsf{OneCompFE.Enc}(\mathsf{msk}'_{i+1}, (f, \widetilde{\mathsf{st}}_{i+1}, r_{\mathsf{msk}_{i+1}}, r_{\mathsf{KeyGen}_{i+1}}, 0, 0^\lambda); r'_{\mathsf{Enc}_{i+1}})$

   iii. $c_i \leftarrow \mathsf{Sym.Enc}(k_i, (\theta_i, \mathsf{ct}'_{i+1}))$

   iv. If $i = 1$

   A. $\mathsf{sk}_{g_1} \leftarrow \mathsf{OneFSFE.KeyGen}(\mathsf{msk}_1, g_{f,\widetilde{\mathsf{st}}_1,c_1})$ for $g_{f,\widetilde{\mathsf{st}}_1,c_1}$ as defined in Figure 6.

   v. If $i > 1$

   A. $c'_i \leftarrow \mathsf{Sym.Enc}(k'_i, 0^{\ell_{\mathsf{Sym'}.m_\lambda}})$

   B. $\mathsf{sk}'_{h_i} \leftarrow \mathsf{OneCompFE.KeyGen}(\mathsf{msk}'_i, h_{c_i,c'_i})$ for h_{c_i,c'_i} as defined in Figure 7.

5. **Function Key:** Send $\mathsf{SK}_f = \mathsf{sk}_{g_1}$ to the adversary.

6. **Challenge Message:** $\mathcal{A}$ outputs a challenge message $x = x_1 \ldots x_n$ for some $n \in \mathbb{N}$ and where each $x_i \in \{0,1\}^{\ell_{\mathcal{X}}}$.

7. **Compute p_i and ψ_i:**

   (a) $\mathsf{st}_1 = 0^{\ell_{\mathcal{S}}}$

   (b) For $i \in [n]$,

   i. $p_i = \widetilde{\mathsf{st}}_i \oplus \mathsf{st}_i$

   ii. $(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$

   iii. $\psi_i = \theta_i \oplus y_i$

   (c) $p_{i+1} = \widetilde{\mathsf{st}}_{i+1} \oplus \mathsf{st}_{i+1}$

8. **Challenge Ciphertext:**

   (a) For $i \in [n]$,

   i. $\mathsf{ct}_i \leftarrow \mathsf{OneFSFE.Enc}(\mathsf{msk}_i, (x_i, p_i, p_{i+1}, r'_{\mathsf{msk}_{i+1}}, r'_{\mathsf{Enc}_{i+1}}, r_{\mathsf{msk}_{i+1}}, r_{\mathsf{KeyGen}_{i+1}}, 0, 0^\lambda, 0^{\ell_{\mathcal{Y}}}))$

ii. If $i = 1$, $\mathsf{CT}_1 = \mathsf{ct}_1$. Else, $\mathsf{CT}_i = (\mathsf{ct}_i, \mathsf{sk}'_{h_i})$

(b) Send $\mathsf{CT} = (\mathsf{CT}_i)_{i\in[n]}$ to the adversary.

9. **Adversary's Output:** $\mathcal{A}$ outputs a bit b which is the outcome of the experiment.

Lemma 5.6. *If* Sym *is a secure symmetric key encryption scheme, then for all PPT adversaries $\mathcal{A}$,*

$$\left|\Pr[\mathbf{Hybrid}_3^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_4^{\mathcal{A}}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

Proof. Suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$ such that

$$\left|\Pr[\mathbf{Hybrid}_3^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_4^{\mathcal{A}}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{3}$$

Let $q = q(\lambda)$ be the running time of $\mathcal{A}$. Observe that $q = \mathsf{poly}(\lambda)$ as $\mathcal{A}$ is polytime and that $q(\lambda) \geq n$ for any challenge message $x = x_1 \ldots x_n$ output by $\mathcal{A}$ on security parameter λ. Let $\mathbf{Hybrid}_{3,j}^{\mathcal{A}}$ be the same as $\mathbf{Hybrid}_3^{\mathcal{A}}$ except that we compute the first j values of c_i as in $\mathbf{Hybrid}_4^{\mathcal{A}}$, i.e.:

$$\theta_i \leftarrow \{0,1\}^{\ell_{\mathcal{Y}}}$$
$$\mathsf{ct}'_{i+1} \leftarrow \mathsf{OneCompFE.Enc}(\mathsf{msk}'_{i+1}, (f, \widetilde{\mathsf{st}}_{i+1}, r_{\mathsf{msk}_{i+1}}, r_{\mathsf{KeyGen}_{i+1}}, 0, 0^\lambda); r'_{\mathsf{Enc}_{i+1}})$$
$$c_i \leftarrow \mathsf{Sym.Enc}(k_i, (\theta_i, \mathsf{ct}'_{i+1}))$$

Observe that $\mathbf{Hybrid}_3^{\mathcal{A}} = \mathbf{Hybrid}_{3,0}^{\mathcal{A}}$ and that $\mathbf{Hybrid}_4^{\mathcal{A}} = \mathbf{Hybrid}_{3,q}^{\mathcal{A}}$. (It does not matter that we do not compute $\{\psi_i\}_{i\in[n]}$ as these values are not used in $\mathbf{Hybrid}_4^{\mathcal{A}}$.) Then, by Equation 3 there must exist a value $j^* \in [q]$ such that

$$\left|\Pr[\mathbf{Hybrid}_{3,j^*-1}^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_{3,j^*}^{\mathcal{A}}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{4}$$

We build a PPT adversary $\mathcal{B}$ that breaks the security of Sym. $\mathcal{B}$ first runs $\mathcal{A}$ on input 1^λ and receives parameters $1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}}$ and a function query $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$. $\mathcal{B}$ then sends message length $1^{\ell_{\mathsf{Sym}.m_\lambda}}$ to its Sym challenger where $\ell_{\mathsf{Sym}.m_\lambda}$ is computed as described in the parameter section. Then, $\mathcal{B}$ samples $\{r_{\mathsf{msk}_i}, r'_{\mathsf{msk}_i}, r'_{k_i}, r'_{\mathsf{Enc}_i}, r_{\mathsf{KeyGen}_i}, \widetilde{\mathsf{st}}_i\}_{i\in[q+1]}$ uniformly at random and computes $\{\mathsf{msk}_i, \mathsf{msk}'_i, k'_i\}_{i\in[n+1]}$ from these values as in $\mathbf{Hybrid}_3^{\mathcal{A}}$. $\mathcal{B}$ also samples $\{r_{k_i}\}_{i\in[q+1]\setminus\{j^*\}}$ uniformly at random and computes $\{k_i\}_{i\in[q+1]\setminus\{j^*\}}$ from these values as in $\mathbf{Hybrid}_3^{\mathcal{A}}$. $\mathcal{B}$ computes $\{\theta_i, \mathsf{ct}'_{i+1}\}_{i\in[q]}$ as in $\mathbf{Hybrid}_4^{\mathcal{A}}$. For $i < j^*$, $\mathcal{B}$ computes c_i as in $\mathbf{Hybrid}_4^{\mathcal{A}}$. For $i > j^*$, $\mathcal{B}$ computes c_i as in $\mathbf{Hybrid}_3^{\mathcal{A}}$. For $i = j^*$, $\mathcal{B}$ sends challenge messages $((\theta_{j^*}, \mathsf{ct}'_{j^*+1}), (0^{\ell_{\mathsf{Sym}.m_\lambda}}))$ to its Sym challenger and receives an encryption c_{j^*} of either $(\theta_{j^*}, \mathsf{ct}'_{j^*+1})$ or $(0^{\ell_{\mathsf{Sym}.m_\lambda}})$. $\mathcal{B}$ then computes $\{c'_i\}_{i\in[q]}, \mathsf{sk}_{g_1}, \{\mathsf{sk}_{h_i}\}_{i\in[q]\setminus\{1\}}$ from these values as in $\mathbf{Hybrid}_3^{\mathcal{A}}$. $\mathcal{B}$ sends $\mathsf{SK}_f = \mathsf{sk}_{g_1}$ to $\mathcal{A}$ and receives a challenge message x. Then, $\mathcal{B}$ computes $\{p_i\}_{i\in[n+1]}$ and CT as in $\mathbf{Hybrid}_3^{\mathcal{A}}$. $\mathcal{B}$ sends CT to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs. Observe that if $\mathcal{B}$ received an encryption c_{j^*} of $(\theta_{j^*}, \mathsf{ct}'_{j^*+1})$ then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}_{4,j}^{\mathcal{A}}$, and if $\mathcal{B}$ received an encryption c_{j^*} of $0^{\ell_{\mathsf{Sym}.m_\lambda}}$ then $\mathcal{B}$ emulates $\mathbf{Hybrid}_{4,j-1}^{\mathcal{A}}$. Additionally, $\mathcal{B}$ does not need to know $r_{k_{j^*}}$ or k_{j^*} to carry out the experiment. Thus, by Equation 4, this means that $\mathcal{B}$ breaks the security of Sym as $\mathcal{B}$ can distinguish between the two ciphertexts with non-negligible probability. $\square$

Hybrid$_5^{\mathcal{A}}(1^\lambda)$: For each i, we hardcode into c'_i the value sk_{g_i} that would be output by $h_i = h_{c_i,c'_i}$ in the $\alpha'_i = 0$ branch if we were to run it on the input generated by the challenge message x. This will allows us to later switch to the $\alpha'_i = 1$ branch in h_{c_i,c'_i} using the security of $\mathsf{OneCompFE}$. Observe that the values being hardcoded into c'_i can be determined before knowing x.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs a function size $1^{\ell_\mathcal{F}}$, a state size $1^{\ell_\mathcal{S}}$, an input size $1^{\ell_\mathcal{X}}$, and an output size $1^{\ell_\mathcal{Y}}$.

2. **Function Query:** $\mathcal{A}$ outputs a streaming function query $f \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$.

3. **Compute Randomness:**

   (a) For $i \in [n+1]$,

   i. $r_{\mathsf{msk}_i}, r'_{\mathsf{msk}_i}, r_{k_i}, r'_{k_i}, r'_{\mathsf{Enc}_i}, r_{\mathsf{KeyGen}_i} \leftarrow \{0,1\}^\lambda$

   ii. $\mathsf{msk}_i \leftarrow \mathsf{OneFSFE.Setup}(1^\lambda; r_{\mathsf{msk}_i})$, $\mathsf{msk}'_i \leftarrow \mathsf{OneCompFE.Setup}(1^\lambda; r'_{\mathsf{msk}_i})$, $k_i \leftarrow \mathsf{Sym.Setup}(1^\lambda; r_{k_i})$, $k'_i \leftarrow \mathsf{Sym'.Setup}(1^\lambda; r'_{k_i})$

   iii. $\widetilde{\mathsf{st}}_i \leftarrow \{0,1\}^{\ell_\mathcal{S}}$

4. **Compute sk_{g_i} and sk'_{h_i}:**

   (a) For $i \in [n]$,

   i. $\theta_i \leftarrow \{0,1\}^{\ell_\mathcal{Y}}$

   ii. $\mathsf{ct}'_{i+1} \leftarrow \mathsf{OneCompFE.Enc}(\mathsf{msk}'_{i+1}, (f, \widetilde{\mathsf{st}}_{i+1}, r_{\mathsf{msk}_{i+1}}, r_{\mathsf{KeyGen}_{i+1}}, 0, 0^\lambda); r'_{\mathsf{Enc}_{i+1}})$

   iii. $c_i \leftarrow \mathsf{Sym.Enc}(k_i, (\theta_i, \mathsf{ct}'_{i+1}))$

   iv. $\mathsf{sk}_{g_i} \leftarrow \mathsf{OneFSFE.KeyGen}(\mathsf{msk}_i, g_{f,\widetilde{\mathsf{st}}_i,c_i}; r_{\mathsf{KeyGen}_i})$ for $g_{f,\widetilde{\mathsf{st}}_i,c_i}$ as defined in Figure 6.

   v. ~~If $i = 1$~~

   A. ~~$\mathsf{sk}_{g_1} \leftarrow \mathsf{OneFSFE.KeyGen}(\mathsf{msk}_1, g_{f,\widetilde{\mathsf{st}}_1,c_1})$ for $g_{f,\widetilde{\mathsf{st}}_1,c_1}$ as defined in Figure 6.~~

   vi. If $i > 1$

   A. $c'_i \leftarrow \mathsf{Sym.Enc}(k'_i, \mathsf{sk}_{g_i})$

   B. $\mathsf{sk}'_{h_i} \leftarrow \mathsf{OneCompFE.KeyGen}(\mathsf{msk}'_i, h_{c_i,c'_i})$ for h_{c_i,c'_i} as defined in Figure 7.

5. **Function Key:** Send $\mathsf{SK}_f = \mathsf{sk}_{g_1}$ to the adversary.

6. **Challenge Message:** $\mathcal{A}$ outputs a challenge message $x = x_1 \dots x_n$ for some $n \in \mathbb{N}$ and where each $x_i \in \{0,1\}^{\ell_\mathcal{X}}$.

7. **Compute p_i and ψ_i:**

   (a) $\mathsf{st}_1 = 0^{\ell_\mathcal{S}}$

   (b) For $i \in [n]$,

   i. $p_i = \widetilde{\mathsf{st}}_i \oplus \mathsf{st}_i$

   ii. $(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$

   iii. $\psi_i = \theta_i \oplus y_i$

   (c) $p_{i+1} = \widetilde{\mathsf{st}}_{i+1} \oplus \mathsf{st}_{i+1}$

8. **Challenge Ciphertext:**

   (a) For $i \in [n]$,

i. $\mathsf{ct}_i \leftarrow \mathsf{OneFSFE.Enc}(\mathsf{msk}_i, (x_i, p_i, p_{i+1}, r'_{\mathsf{msk}_{i+1}}, r'_{\mathsf{Enc}_{i+1}}, r_{\mathsf{msk}_{i+1}}, r_{\mathsf{KeyGen}_{i+1}}, 0, 0^\lambda, 0^{\ell_\mathcal{Y}}))$

ii. If $i = 1$, $\mathsf{CT}_1 = \mathsf{ct}_1$. Else, $\mathsf{CT}_i = (\mathsf{ct}_i, \mathsf{sk}'_{h_i})$

(b) Send $\mathsf{CT} = (\mathsf{CT}_i)_{i\in[n]}$ to the adversary.

9. **Adversary's Output:** $\mathcal{A}$ outputs a bit b which is the outcome of the experiment.

Lemma 5.7. *If* Sym' *is a secure symmetric key encryption scheme, then for all PPT adversaries* $\mathcal{A}$*,*

$$\left|\Pr[\mathbf{Hybrid}_4^\mathcal{A}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_5^\mathcal{A}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

Proof. Suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$ such that

$$\left|\Pr[\mathbf{Hybrid}_4^\mathcal{A}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_5^\mathcal{A}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{5}$$

Let $q = q(\lambda)$ be the running time of $\mathcal{A}$. Observe that $q = \mathsf{poly}(\lambda)$ as $\mathcal{A}$ is polytime and that $q(\lambda) \geq n$ for any challenge message $x = x_1 \dots x_n$ output by $\mathcal{A}$ on security parameter λ. Let $\mathbf{Hybrid}_{4,j}^\mathcal{A}$ be the same as $\mathbf{Hybrid}_4^\mathcal{A}$ except that we compute the values of c'_i for $i \in [j]\backslash\{1\}$ as in $\mathbf{Hybrid}_5^\mathcal{A}$, i.e.:

$$\mathsf{sk}_{g_i} \leftarrow \mathsf{OneFSFE.KeyGen}(\mathsf{msk}_i, g_{f,\widetilde{\mathsf{st}}_i,c_i}; r_{\mathsf{KeyGen}_i})$$
$$c'_i \leftarrow \mathsf{Sym.Enc}(k'_i, \mathsf{sk}_{g_i})$$

Observe that $\mathbf{Hybrid}_4^\mathcal{A} = \mathbf{Hybrid}_{4,0}^\mathcal{A}$ and that $\mathbf{Hybrid}_5^\mathcal{A} = \mathbf{Hybrid}_{4,q}^\mathcal{A}$. Then by Equation 5, there must exist a value $j^* \in [q]$ such that

$$\left|\Pr[\mathbf{Hybrid}_{4,j^*-1}^\mathcal{A}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_{4,j^*}^\mathcal{A}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{6}$$

We build a PPT adversary $\mathcal{B}$ that breaks the security of Sym'. $\mathcal{B}$ first runs $\mathcal{A}$ on input 1^λ and receives parameters $1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{S}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}}$ and a function query $f \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$. $\mathcal{B}$ then sends message length $1^{\ell_{\mathsf{Sym}'.m_\lambda}}$ to its Sym' challenger where $\ell_{\mathsf{Sym}'.m_\lambda}$ is computed as described in the parameter section. Then, $\mathcal{B}$ samples $\{r_{\mathsf{msk}_i}, r'_{\mathsf{msk}_i}, r_{k_i}, r'_{\mathsf{Enc}_i}, r_{\mathsf{KeyGen}_i}, \widetilde{\mathsf{st}}_i\}_{i\in[q+1]}$ uniformly at random and computes $\{\mathsf{msk}_i, \mathsf{msk}'_i, k_i\}_{i\in[q+1]}$ from these values as in $\mathbf{Hybrid}_4^\mathcal{A}$. $\mathcal{B}$ also samples $\{r'_{k_i}\}_{i\in[q+1]\backslash\{j^*\}}$ uniformly at random and computes $\{k'_i\}_{i\in[q+1]\backslash\{j^*\}}$ from these values as in $\mathbf{Hybrid}_4^\mathcal{A}$. $\mathcal{B}$ computes $\{c_i\}_{i\in[q]}$ as in $\mathbf{Hybrid}_4^\mathcal{A}$, and computes $sk_{g_i} \leftarrow \mathsf{OneFSFE.KeyGen}(\mathsf{msk}_i, g_{f,\widetilde{\mathsf{st}}_i,c_i}; r_{\mathsf{KeyGen}_i})$ for $i \in [q]$ as in $\mathbf{Hybrid}_5^\mathcal{A}$. For $i < j^*$, $\mathcal{B}$ computes c'_i as in $\mathbf{Hybrid}_5^\mathcal{A}$. For $i > j^*$, $\mathcal{B}$ computes c'_i as in $\mathbf{Hybrid}_4^\mathcal{A}$. For $i = j^*$, $\mathcal{B}$ sends challenge messages $(\mathsf{sk}_{g_{j^*}}, 0^{\ell_{\mathsf{Sym}'.m_\lambda}})$ to its Sym' challenger and receives an encryption c'_{j^*} of either $\mathsf{sk}_{g_{j^*}}$ or $0^{\ell_{\mathsf{Sym}'.m_\lambda}}$. $\mathcal{B}$ then computes $\{\mathsf{sk}'_{h_i}\}_{i\in[q]\backslash\{1\}}$ as in $\mathbf{Hybrid}_4^\mathcal{A}$. $\mathcal{B}$ sends $\mathsf{SK}_f = \mathsf{sk}_{g_1}$ to $\mathcal{A}$ and receives a challenge message x. Then, $\mathcal{B}$ computes $\{p_i\}_{i\in[n+1]}, \{\psi_i\}_{i\in[n]}$ and CT as in $\mathbf{Hybrid}_4^\mathcal{A}$, sends CT to $\mathcal{A}$, and outputs whatever $\mathcal{A}$ outputs. Observe that if $\mathcal{B}$ received an encryption c'_{j^*} of $\mathsf{sk}_{g_{j^*}}$ then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}_{4,j^*}^\mathcal{A}$, and if $\mathcal{B}$ received an encryption c'_{j^*} of $0^{\ell_{\mathsf{Sym}'.m_\lambda}}$ then $\mathcal{B}$ emulates $\mathbf{Hybrid}_{4,j^*-1}^\mathcal{A}$. Additionally, $\mathcal{B}$ does not need to know $r'_{k_{j^*}}$ or k'_{j^*} to carry out the experiment. Thus, by Equation 6, this means that $\mathcal{B}$ breaks the security of Sym' as $\mathcal{B}$ can distinguish between the two ciphertexts with non-negligible probability. □

Hybrid$^{\mathcal{A}}_{6,k,1}(1^\lambda)$: We change the message encrypted in ct_k so that we use the $\alpha_k = 1$ branch of $g_{f,\widetilde{\mathsf{st}}_k,c_k}$.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs a function size $1^{\ell_\mathcal{F}}$, a state size $1^{\ell_\mathcal{S}}$, an input size $1^{\ell_\mathcal{X}}$, and an output size $1^{\ell_\mathcal{Y}}$.
2. **Function Query:** $\mathcal{A}$ outputs a streaming function query $f \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$.
3. **Compute Randomness:**
   (a) For $i \in [n+1]$,
      i. $r_{\mathsf{msk}_i}, r'_{\mathsf{msk}_i}, r_{k_i}, r'_{k_i}, r'_{\mathsf{Enc}_i}, r_{\mathsf{KeyGen}_i} \leftarrow \{0,1\}^\lambda$
      ii. $\mathsf{msk}_i \leftarrow \mathsf{OneFSFE.Setup}(1^\lambda; r_{\mathsf{msk}_i})$, $\mathsf{msk}'_i \leftarrow \mathsf{OneCompFE.Setup}(1^\lambda; r'_{\mathsf{msk}_i})$, $k_i \leftarrow \mathsf{Sym.Setup}(1^\lambda; r_{k_i})$, $k'_i \leftarrow \mathsf{Sym'.Setup}(1^\lambda; r'_{k_i})$
      iii. $\widetilde{\mathsf{st}}_i \leftarrow \{0,1\}^{\ell_\mathcal{S}}$
4. **Compute sk_{g_i} and sk'_{h_i}:**
   (a) For $i \in [n]$,
      i. $\theta_i \leftarrow \{0,1\}^{\ell_\mathcal{Y}}$
      ii. If $i < k, \mathsf{ct}'_{i+1} \leftarrow \mathsf{OneCompFE.Enc}(\mathsf{msk}'_{i+1}, (0^{\ell_\mathcal{F}}, 0^{\ell_\mathcal{S}}, 0^\lambda, 0^\lambda, 1, r'_{k_i}); r'_{\mathsf{Enc}_{i+1}})$
      iii. If $i \geq k, \mathsf{ct}'_{i+1} \leftarrow \mathsf{OneCompFE.Enc}(\mathsf{msk}'_{i+1}, (f, \widetilde{\mathsf{st}}_{i+1}, r_{\mathsf{msk}_{i+1}}, r_{\mathsf{KeyGen}_{i+1}}, 0, 0^\lambda); r'_{\mathsf{Enc}_{i+1}})$
      iv. $c_i \leftarrow \mathsf{Sym.Enc}(k_i, (\theta_i, \mathsf{ct}'_{i+1}))$
      v. $\mathsf{sk}_{g_i} \leftarrow \mathsf{OneFSFE.KeyGen}(\mathsf{msk}_i, g_{f,\widetilde{\mathsf{st}}_i,c_i}; r_{\mathsf{KeyGen}_i})$ for $g_{f,\widetilde{\mathsf{st}}_i,c_i}$ as defined in Figure 6.
      vi. If $i > 1$
         A. $c'_i \leftarrow \mathsf{Sym.Enc}(k'_i, \mathsf{sk}_{g_i})$
         B. $\mathsf{sk}'_{h_i} \leftarrow \mathsf{OneCompFE.KeyGen}(\mathsf{msk}'_i, h_{c_i,c'_i})$ for h_{c_i,c'_i} as defined in Figure 7.
5. **Function Key:** Send $\mathsf{SK}_f = \mathsf{sk}_{g_1}$ to the adversary.
6. **Challenge Message:** $\mathcal{A}$ outputs a challenge message $x = x_1 \ldots x_n$ for some $n \in \mathbb{N}$ and where each $x_i \in \{0,1\}^{\ell_\mathcal{X}}$.
7. **Compute p_i and ψ_i:**
   (a) $\mathsf{st}_1 = 0^{\ell_\mathcal{S}}$
   (b) For $i \in [n]$,
      i. $p_i = \widetilde{\mathsf{st}}_i \oplus \mathsf{st}_i$
      ii. $(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$
      iii. $\psi_i = \theta_i \oplus y_i$
   (c) $p_{i+1} = \widetilde{\mathsf{st}}_{i+1} \oplus \mathsf{st}_{i+1}$
8. **Challenge Ciphertext:**
   (a) For $i \in [n]$,
      i. If $i \leq k, \mathsf{ct}_i \leftarrow \mathsf{OneFSFE.Enc}(\mathsf{msk}_i, (0^{\ell_\mathcal{X}}, 0^{\ell_\mathcal{S}}, 0^{\ell_\mathcal{S}}, 0^\lambda, 0^\lambda, 0^\lambda, 0^\lambda, 1, r_{k_i}, \psi_i))$
      ii. If $i > k, \mathsf{ct}_i \leftarrow \mathsf{OneFSFE.Enc}(\mathsf{msk}_i, (x_i, p_i, p_{i+1}, r'_{\mathsf{msk}_{i+1}}, r'_{\mathsf{Enc}_{i+1}}, r_{\mathsf{msk}_{i+1}}, r_{\mathsf{KeyGen}_{i+1}}, 0, 0^\lambda, 0^{\ell_\mathcal{Y}}))$
      iii. If $i = 1$, $\mathsf{CT}_1 = \mathsf{ct}_1$. Else, $\mathsf{CT}_i = (\mathsf{ct}_i, \mathsf{sk}'_{h_i})$
   (b) Send $\mathsf{CT} = (\mathsf{CT}_i)_{i \in [n]}$ to the adversary.
9. **Adversary's Output:** $\mathcal{A}$ outputs a bit b which is the outcome of the experiment.

Hybrid$^{\mathcal{A}}_{6,k,2}(1^\lambda)$: We change the message encrypted in ct'_{k+1} so that we use the $\alpha'_{k+1} = 1$ branch of $h_{c_{k+1},c'_{k+1}}$.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs a function size $1^{\ell_\mathcal{F}}$, a state size $1^{\ell_\mathcal{S}}$, an input size $1^{\ell_\mathcal{X}}$, and an output size $1^{\ell_\mathcal{Y}}$.
2. **Function Query:** $\mathcal{A}$ outputs a streaming function query $f \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$.
3. **Compute Randomness:**
   (a) For $i \in [n+1]$,
      i. $r_{\mathsf{msk}_i}, r'_{\mathsf{msk}_i}, r_{k_i}, r'_{k_i}, r'_{\mathsf{Enc}_i}, r_{\mathsf{KeyGen}_i} \leftarrow \{0,1\}^\lambda$
      ii. $\mathsf{msk}_i \leftarrow \mathsf{OneFSFE.Setup}(1^\lambda; r_{\mathsf{msk}_i})$, $\mathsf{msk}'_i \leftarrow \mathsf{OneCompFE.Setup}(1^\lambda; r'_{\mathsf{msk}_i})$, $k_i \leftarrow \mathsf{Sym.Setup}(1^\lambda; r_{k_i})$, $k'_i \leftarrow \mathsf{Sym'.Setup}(1^\lambda; r'_{k_i})$
      iii. $\widetilde{\mathsf{st}}_i \leftarrow \{0,1\}^{\ell_\mathcal{S}}$
4. **Compute sk_{g_i} and sk'_{h_i}:**
   (a) For $i \in [n]$,
      i. $\theta_i \leftarrow \{0,1\}^{\ell_\mathcal{Y}}$
      ii. If $i \le k, \mathsf{ct}'_{i+1} \leftarrow \mathsf{OneCompFE.Enc}(\mathsf{msk}'_{i+1}, (0^{\ell_\mathcal{F}}, 0^{\ell_\mathcal{S}}, 0^\lambda, 0^\lambda, 1, r'_{k_i}); r'_{\mathsf{Enc}_{i+1}})$
      iii. If $i > k, \mathsf{ct}'_{i+1} \leftarrow \mathsf{OneCompFE.Enc}(\mathsf{msk}'_{i+1}, (f, \widetilde{\mathsf{st}}_{i+1}, r_{\mathsf{msk}_{i+1}}, r_{\mathsf{KeyGen}_{i+1}}, 0, 0^\lambda); r'_{\mathsf{Enc}_{i+1}})$
      iv. $c_i \leftarrow \mathsf{Sym.Enc}(k_i, (\theta_i, \mathsf{ct}'_{i+1}))$
      v. $\mathsf{sk}_{g_i} \leftarrow \mathsf{OneFSFE.KeyGen}(\mathsf{msk}_i, g_{f,\widetilde{\mathsf{st}}_i,c_i}; r_{\mathsf{KeyGen}_i})$ for $g_{f,\widetilde{\mathsf{st}}_i,c_i}$ as defined in Figure 6.
      vi. If $i > 1$
         A. $c'_i \leftarrow \mathsf{Sym.Enc}(k'_i, \mathsf{sk}_{g_i})$
         B. $\mathsf{sk}'_{h_i} \leftarrow \mathsf{OneCompFE.KeyGen}(\mathsf{msk}'_i, h_{c_i,c'_i})$ for h_{c_i,c'_i} as defined in Figure 7.
5. **Function Key:** Send $\mathsf{SK}_f = \mathsf{sk}_{g_1}$ to the adversary.
6. **Challenge Message:** $\mathcal{A}$ outputs a challenge message $x = x_1 \dots x_n$ for some $n \in \mathbb{N}$ and where each $x_i \in \{0,1\}^{\ell_\mathcal{X}}$.
7. **Compute p_i and ψ_i:**
   (a) $\mathsf{st}_1 = 0^{\ell_\mathcal{S}}$
   (b) For $i \in [n]$,
      i. $p_i = \widetilde{\mathsf{st}}_i \oplus \mathsf{st}_i$
      ii. $(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$
      iii. $\psi_i = \theta_i \oplus y_i$
   (c) $p_{i+1} = \widetilde{\mathsf{st}}_{i+1} \oplus \mathsf{st}_{i+1}$
8. **Challenge Ciphertext:**
   (a) For $i \in [n]$,
      i. If $i \le k, \mathsf{ct}_i \leftarrow \mathsf{OneFSFE.Enc}(\mathsf{msk}_i, (0^{\ell_\mathcal{X}}, 0^{\ell_\mathcal{S}}, 0^{\ell_\mathcal{S}}, 0^\lambda, 0^\lambda, 0^\lambda, 0^\lambda, 1, r_{k_i}, \psi_i))$
      ii. If $i > k, \mathsf{ct}_i \leftarrow \mathsf{OneFSFE.Enc}(\mathsf{msk}_i, (x_i, p_i, p_{i+1}, r'_{\mathsf{msk}_{i+1}}, r'_{\mathsf{Enc}_{i+1}}, r_{\mathsf{msk}_{i+1}}, r_{\mathsf{KeyGen}_{i+1}}, 0, 0^\lambda, 0^{\ell_\mathcal{S}}))$
      iii. If $i = 1$, $\mathsf{CT}_1 = \mathsf{ct}_1$. Else, $\mathsf{CT}_i = (\mathsf{ct}_i, \mathsf{sk}'_{h_i})$

(b) Send $\mathsf{CT} = (\mathsf{CT}_i)_{i\in[n]}$ to the adversary.

9. **Adversary's Output:** $\mathcal{A}$ outputs a bit b which is the outcome of the experiment.

Lemma 5.8. *For all adversaries $\mathcal{A}$,*

$$\left|\Pr[\mathbf{Hybrid}_5^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_{6,0,2}^{\mathcal{A}}(1^\lambda) = 1]\right| = 0$$

Proof. The hybrids are identical. □

Lemma 5.9. *If* $\mathsf{OneFSFE}$ *is a single-key, single-ciphertext, function-selective-IND-secure FE scheme, then for all PPT adversaries $\mathcal{A}$ and for all $k \in \mathbb{N}$,*

$$\left|\Pr[\mathbf{Hybrid}_{6,k-1,2}^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_{6,k,1}^{\mathcal{A}}(1^\lambda) = 1]\right| \le \mathsf{negl}(\lambda)$$

Proof. Suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$ such that

$$\left|\Pr[\mathbf{Hybrid}_{6,k-1,2}^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_{6,k,1}^{\mathcal{A}}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{7}$$

Let $q = q(\lambda)$ be the running time of $\mathcal{A}$. Observe that $q = \mathsf{poly}(\lambda)$ as $\mathcal{A}$ is polytime and that $q(\lambda) \ge n$ for any challenge message $x = x_1 \dots x_n$ output by $\mathcal{A}$ on security parameter λ. We build a PPT adversary $\mathcal{B}$ that breaks the security of $\mathsf{OneFSFE}$. $\mathcal{B}$ first runs $\mathcal{A}$ on input 1^λ and receives parameters $1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{S}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}}$ and a function query $f \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$. $\mathcal{B}$ then sends function size $1^{\ell_{g_\lambda}}$, input size $1^{\ell_{m_\lambda}}$, and output size $1^{\ell_{\mathsf{Sym}.m_\lambda}}$ to its $\mathsf{OneFSFE}$ challenger where $\ell_{g_\lambda}, \ell_{m_\lambda}, \ell_{\mathsf{Sym}.m_\lambda}$ are computed as specified in our parameter section. $\mathcal{B}$ then samples $\{r'_{\mathsf{msk}_i}, r_{k_i}, r'_{k_i}, r'_{\mathsf{Enc}_i}, \widetilde{\mathsf{st}}_i, \theta_i\}_{i\in[q+1]}$ uniformly at random and computes $\{\mathsf{msk}'_i, k_i, k'_i\}_{i\in[n+1]}$ from these values as in $\mathbf{Hybrid}_{6,k-1,2}^{\mathcal{A}}$. $\mathcal{B}$ also samples $\{r_{\mathsf{msk}_i}, r_{\mathsf{KeyGen}_i}\}_{i\in[q+1]\setminus\{k\}}$, uniformly at random and computes $\{\mathsf{msk}_i\}_{i\in[q+1]\setminus\{k\}}$ from these values as in $\mathbf{Hybrid}_{6,k-1,2}^{\mathcal{A}}$. For $i < k$, $\mathcal{B}$ computes $\mathsf{ct}'_{i+1} \leftarrow \mathsf{OneCompFE.Enc}(\mathsf{msk}'_{i+1}, (0^{\ell_\mathcal{F}}, 0^{\ell_\mathcal{S}}, 0^\lambda, 0^\lambda, 1, r'_{k_i}); r'_{\mathsf{Enc}_{i+1}})$. For $i \ge k$, $\mathcal{B}$ computes $\mathsf{ct}'_{i+1} \leftarrow \mathsf{OneCompFE.Enc}(\mathsf{msk}'_{i+1}, (f, \widetilde{\mathsf{st}}_{i+1}, r_{\mathsf{msk}_{i+1}}, r_{\mathsf{KeyGen}_{i+1}}, 0, 0^\lambda); r'_{\mathsf{Enc}_{i+1}})$. Observe that this is the same as in $\mathbf{Hybrid}_{6,k-1,2}^{\mathcal{A}}$ and does not require knowledge of $\mathsf{msk}_k, r_{\mathsf{msk}_k}, r_{\mathsf{KeyGen}_k}$. $\mathcal{B}$ computes $\{c_i, g_{f,\widetilde{\mathsf{st}}_i,c_i}\}_{i\in[q]}$ from these values as in $\mathbf{Hybrid}_{6,k-1,2}^{\mathcal{A}}$. $\mathcal{B}$ then sends function query $g_k = g_{f,\widetilde{\mathsf{st}}_k,c_k}$ to its $\mathsf{OneFSFE}$ challenger and receives a $\mathsf{OneFSFE}$ function key sk_{g_k} in return. $\mathcal{B}$ computes $\{\mathsf{sk}_{g_i}\}_{i\in[q]\setminus\{k\}}$ and $\{c'_i, \mathsf{sk}'_{h_i}\}_{i\in[q]\setminus\{1\}}$ as in $\mathbf{Hybrid}_{6,k-1,2}^{\mathcal{A}}$. (This does not require knowledge of $\mathsf{msk}_k, r_{\mathsf{msk}_k}, r_{\mathsf{KeyGen}_k}$). $\mathcal{B}$ sends $\mathsf{SK}_f = \mathsf{sk}_{g_1}$ to $\mathcal{A}$ and receives a challenge message x. $\mathcal{B}$ then computes $\{p_i\}_{i\in[n+1]}, \{\psi_i\}_{i\in[n]}$ as in $\mathbf{Hybrid}_{6,k-1,2}^{\mathcal{A}}$. For $i \le k$, let $m_{0,i} = (0^{\ell_\mathcal{X}}, 0^{\ell_\mathcal{S}}, 0^{\ell_\mathcal{S}}, 0^\lambda, 0^\lambda, 0^\lambda, 0^\lambda, 1, r_{k_i}, \psi_i)$, and for $i \ge k$, let $m_{1,i} = (x_i, p_i, p_{i+1}, r'_{\mathsf{msk}_{i+1}}, r'_{\mathsf{Enc}_{i+1}}, r_{\mathsf{msk}_{i+1}}, r_{\mathsf{KeyGen}_{i+1}}, 0, 0^\lambda, 0^{\ell_\mathcal{Y}})$. For $i < k$, $\mathcal{B}$ computes $\mathsf{ct}_i \leftarrow \mathsf{OneFSFE.Enc}(\mathsf{msk}_i, m_{0,i})$. For $i > k$, $\mathcal{B}$ computes $\mathsf{ct}_i \leftarrow \mathsf{OneFSFE.Enc}(\mathsf{msk}_i, m_{1,i})$. Observe that this is the same as in $\mathbf{Hybrid}_{6,k-1,2}^{\mathcal{A}}$ and does not require knowledge of $\mathsf{msk}_k, r_{\mathsf{msk}_k}, r_{\mathsf{KeyGen}_k}$. For $i = k$, $\mathcal{B}$ sends challenge message pair $(m_{0,k}, m_{1,k})$ to its $\mathsf{OneFSFE}$ challenger and receives a $\mathsf{OneFSFE}$ ciphertext ct_k of either $m_{0,k}$ or $m_{1,k}$. As needed for the security game, we can observe that

$$\begin{aligned}
g_k(m_{0,k}) &= g_{f,\widetilde{\mathsf{st}}_k,c_k}(0^{\ell_\mathcal{X}}, 0^{\ell_\mathcal{S}}, 0^{\ell_\mathcal{S}}, 0^\lambda, 0^\lambda, 0^\lambda, 0^\lambda, 1, r_{k_k}, \psi_k)\\
=g_k(m_{1,k}) &= g_{f,\widetilde{\mathsf{st}}_k,c_k}(x_k, p_k, p_{k+1}, r'_{\mathsf{msk}_{k+1}}, r'_{\mathsf{Enc}_{k+1}}, r_{\mathsf{msk}_{k+1}}, r_{\mathsf{KeyGen}_{k+1}}, 0, 0^\lambda, 0^{\ell_\mathcal{Y}})
\end{aligned}$$

This is because c_k encrypts $(\theta_k, \mathsf{ct}'_{k+1})$ where $\theta_k \oplus \psi_k = y_k$ and where ct'_{k+1} is generated in the same way as in the $\alpha_k = 0$ branch of g_k. $\mathcal{B}$ sets $\mathsf{CT}_1 = \mathsf{ct}_1$ and $\mathsf{CT}_i = (\mathsf{ct}_i, \mathsf{sk}'_{h_i})$ for $i \in [n]\setminus\{1\}$. $\mathcal{B}$ sends

$\mathsf{CT} = (\mathsf{CT}_i)_{i\in[n]}$ to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs. Observe that if $\mathcal{B}$ received an encryption ct_k of $m_{0,k}$ then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,1}$, and if $\mathcal{B}$ received an encryption ct_k of $m_{1,k}$ then $\mathcal{B}$ emulates $\mathbf{Hybrid}^{\mathcal{A}}_{6,k-1,2}$. Additionally, $\mathcal{B}$ does not need to know the values of $\mathsf{msk}_k, r_{\mathsf{msk}_k}, r_{\mathsf{KeyGen}_k}$ to run this experiment. Thus, by Equation 7, $\mathcal{B}$ breaks the security of $\mathsf{OneFSFE}$ as $\mathcal{B}$ can distinguish between the two ciphertexts with non-negligible probability. $\square$

Lemma 5.10. *If* $\mathsf{OneCompFE}$ *is a single-key, single-ciphertext, selective-IND-secure FE scheme, then for all PPT adversaries* $\mathcal{A}$ *and for all* $k \in \mathbb{N}$,

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,1}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,2}(1^\lambda) = 1]\right| \le \mathsf{negl}(\lambda)$$

Proof. Suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$ such that

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,1}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,2}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{8}$$

Let $q = q(\lambda)$ be the running time of $\mathcal{A}$. Observe that $q = \mathsf{poly}(\lambda)$ as $\mathcal{A}$ is polytime and that $q(\lambda) \ge n$ for any challenge message $x = x_1 \dots x_n$ output by $\mathcal{A}$ on security parameter λ. We build a PPT adversary $\mathcal{B}$ that breaks the single-key, single-ciphertext, selective-IND-security of $\mathsf{OneCompFE}$. $\mathcal{B}$ first runs $\mathcal{A}$ on input 1^λ and receives parameters $1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{S}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}}$ and a function query $f \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$. $\mathcal{B}$ then sends function size $1^{\ell_{h_\lambda}}$, input size $1^{\ell_{m'_\lambda}}$, and output size $1^{\ell_{\mathsf{Sym}'.m_\lambda}}$ to its $\mathsf{OneCompFE}$ challenger where $\ell_{h_\lambda}, \ell_{m'_\lambda}, \ell_{\mathsf{Sym}'.m_\lambda}$ are computed as specified in our parameter section. $\mathcal{B}$ then samples $\{r_{\mathsf{msk}_i}, r_{k_i}, r'_{k_i}, r_{\mathsf{KeyGen}_i}, \widetilde{\mathsf{st}}_i, \theta_i\}_{i\in[q+1]}$ uniformly at random and computes $\{\mathsf{msk}_i, k_i, k'_i\}_{i\in[q+1]}$ from these values as in $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,1}$. $\mathcal{B}$ also samples $\{r'_{\mathsf{msk}_i}, r'_{\mathsf{Enc}_i}\}_{i\in[q+1]\setminus\{k+1\}}$, uniformly at random and computes $\{\mathsf{msk}'_i\}_{i\in[q+1]\setminus\{k+1\}}$ from these values as in $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,1}$. For $i < k$, $\mathcal{B}$ computes $\mathsf{ct}'_{i+1} \leftarrow \mathsf{OneCompFE.Enc}(\mathsf{msk}'_{i+1}, (0^{\ell_\mathcal{F}}, 0^{\ell_\mathcal{S}}, 0^\lambda, 0^\lambda, 1, r'_{k_i}); r'_{\mathsf{Enc}_{i+1}})$. For $i > k$, $\mathcal{B}$ computes $\mathsf{ct}'_{i+1} \leftarrow \mathsf{OneCompFE.Enc}(\mathsf{msk}'_{i+1}, (f, \widetilde{\mathsf{st}}_{i+1}, r_{\mathsf{msk}_{i+1}}, r_{\mathsf{KeyGen}_{i+1}}, 0, 0^\lambda); r'_{\mathsf{Enc}_{i+1}})$. Observe that this is the same as in $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,1}$ and does not require knowledge of $\mathsf{msk}'_{k+1}, r'_{\mathsf{msk}_{k+1}}, r'_{\mathsf{Enc}_{k+1}}$. For $i = k$, $\mathcal{B}$ sends challenge message pair $((0^{\ell_\mathcal{F}}, 0^{\ell_\mathcal{S}}, 0^\lambda, 0^\lambda, 1, r'_{k_k}), (f, \widetilde{\mathsf{st}}_{k+1}, r_{\mathsf{msk}_{k+1}}, r_{\mathsf{KeyGen}_{k+1}}, 0, 0^\lambda))$ to its $\mathsf{OneCompFE}$ challenger and receives a $\mathsf{OneCompFE}$ encryption ct'_{k+1} of either $(0^{\ell_\mathcal{F}}, 0^{\ell_\mathcal{S}}, 0^\lambda, 0^\lambda, 1, r'_{k_k})$ or $(f, \widetilde{\mathsf{st}}_{k+1}, r_{\mathsf{msk}_{k+1}}, r_{\mathsf{KeyGen}_{k+1}}, 0, 0^\lambda)$. $\mathcal{B}$ computes $\{c_i, \mathsf{sk}_{g_i}\}_{i\in[q]}, \{c'_i, h_{c_i,c'_i}\}_{i\in[q]\setminus\{1\}}$ from these values as in $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,1}$. (This does not require knowledge of $\mathsf{msk}'_{k+1}, r'_{\mathsf{msk}_{k+1}}, r'_{\mathsf{Enc}_{k+1}}$). $\mathcal{B}$ then sends function query $h_{k+1} = h_{c_{k+1},c'_{k+1}}$ to its $\mathsf{OneCompFE}$ challenger and receives a $\mathsf{OneCompFE}$ function key $\mathsf{sk}'_{h_{k+1}}$ in return. As needed for the security game, we can observe that

$$\begin{aligned} &h_{c_{k+1},c'_{k+1}}(0^{\ell_\mathcal{F}}, 0^{\ell_\mathcal{S}}, 0^\lambda, 0^\lambda, 1, r'_{k_k}) \\ &= h_{c_{k+1},c'_{k+1}}(f, \widetilde{\mathsf{st}}_{k+1}, r_{\mathsf{msk}_{k+1}}, r_{\mathsf{KeyGen}_{k+1}}, 0, 0^\lambda) \end{aligned}$$

This is because c'_{k+1} encrypts $\mathsf{sk}_{g_{k+1}}$ where $\mathsf{sk}_{g_{k+1}}$ is generated in the same way as in the $\alpha'_{k+1} = 0$ branch of h_{k+1}. $\mathcal{B}$ computes $\{\mathsf{sk}'_{h_i}\}_{i\in[q]\setminus\{1,k+1\}}$ as in $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,1}$. $\mathcal{B}$ sends $\mathsf{SK}_f = \mathsf{sk}_{g_1}$ to $\mathcal{A}$ and receives a challenge message x. $\mathcal{B}$ then computes $\{p_i\}_{i\in[n+1]}, \{\psi_i\}_{i\in[n]}$ as in $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,1}$. For $i \le k$, $\mathcal{B}$ computes $\mathsf{ct}_i \leftarrow \mathsf{OneCompFE.Enc}(\mathsf{msk}_i, (0^{\ell_\mathcal{X}}, 0^{\ell_\mathcal{S}}, 0^{\ell_\mathcal{S}}, 0^\lambda, 0^\lambda, 0^\lambda, 0^\lambda, 1, r_{k_i}, \psi_i))$. For $i > k$, $\mathcal{B}$ computes $\mathsf{ct}_i \leftarrow \mathsf{OneCompFE.Enc}(\mathsf{msk}_i, (x_i, p_i, p_{i+1}, r'_{\mathsf{msk}_{i+1}}, r'_{\mathsf{Enc}_{i+1}}, r_{\mathsf{msk}_{i+1}}, r_{\mathsf{KeyGen}_{i+1}}, 0, 0^\lambda, 0^{\ell_\mathcal{Y}}))$. Observe that this is the same as in $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,1}$ and does not require knowledge of $\mathsf{msk}'_{k+1}, r'_{\mathsf{msk}_{k+1}}, r'_{\mathsf{Enc}_{k+1}}$. $\mathcal{B}$ sets $\mathsf{CT}_1 = \mathsf{ct}_1$ and $\mathsf{CT}_i = (\mathsf{ct}_i, \mathsf{sk}'_{h_i})$ for $i \in [n]\setminus\{1\}$. $\mathcal{B}$ sends $\mathsf{CT} = (\mathsf{CT}_i)_{i\in[n]}$ to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs. Observe that if $\mathcal{B}$ received an encryption ct'_{k+1} of $(0^{\ell_\mathcal{F}}, 0^{\ell_\mathcal{S}}, 0^\lambda, 0^\lambda, 1, r'_{k_k})$ then $\mathcal{B}$

exactly emulates $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,2}$, and if $\mathcal{B}$ received an encryption ct'_{k+1} of $(f, \widetilde{\mathsf{st}}_{k+1}, r_{\mathsf{msk}_{k+1}}, r_{\mathsf{KeyGen}_{k+1}}, 0, 0^\lambda)$ then $\mathcal{B}$ emulates $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,1}$. Additionally, $\mathcal{B}$ does not need to know the values of $\mathsf{msk}'_{k+1}, r'_{\mathsf{msk}_{k+1}}, r'_{\mathsf{Enc}_{k+1}}$ to run this experiment. Thus, by Equation 8, $\mathcal{B}$ breaks the security of $\mathsf{OneCompFE}$ as $\mathcal{B}$ can distinguish between the two ciphertexts with non-negligible probability. $\square$

Hybrid$_7^{\mathcal{A}}(1^\lambda)$: For any $\mathcal{A}$, this is simply **Hybrid**$_{6,q,2}^{\mathcal{A}}$ where $q = q(\lambda)$ is the runtime of $\mathcal{A}$ on security parameter λ.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs a function size $1^{\ell_\mathcal{F}}$, a state size $1^{\ell_\mathcal{S}}$, an input size $1^{\ell_\mathcal{X}}$, and an output size $1^{\ell_\mathcal{Y}}$.
2. **Function Query:** $\mathcal{A}$ outputs a streaming function query $f \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$.
3. **Compute Randomness:**
   (a) For $i \in [n+1]$,
      i. $r_{\mathsf{msk}_i}, r'_{\mathsf{msk}_i}, r_{k_i}, r'_{k_i}, r'_{\mathsf{Enc}_i}, r_{\mathsf{KeyGen}_i} \leftarrow \{0,1\}^\lambda$
      ii. $\mathsf{msk}_i \leftarrow \mathsf{OneFSFE.Setup}(1^\lambda; r_{\mathsf{msk}_i})$, $\mathsf{msk}'_i \leftarrow \mathsf{OneCompFE.Setup}(1^\lambda; r'_{\mathsf{msk}_i})$, $k_i \leftarrow \mathsf{Sym.Setup}(1^\lambda; r_{k_i})$, $k'_i \leftarrow \mathsf{Sym}'.\mathsf{Setup}(1^\lambda; r'_{k_i})$
      iii. $\widetilde{\mathsf{st}}_i \leftarrow \{0,1\}^{\ell_\mathcal{S}}$
4. **Compute sk_{g_i} and sk'_{h_i}:**
   (a) For $i \in [n]$,
      i. $\theta_i \leftarrow \{0,1\}^{\ell_\mathcal{Y}}$
      ii. $\mathsf{ct}'_{i+1} \leftarrow \mathsf{OneCompFE.Enc}(\mathsf{msk}'_{i+1}, (0^{\ell_\mathcal{F}}, 0^{\ell_\mathcal{S}}, 0^\lambda, 0^\lambda, 1, r'_{k_i}); r'_{\mathsf{Enc}_{i+1}})$
      iii. $c_i \leftarrow \mathsf{Sym.Enc}(k_i, (\theta_i, \mathsf{ct}'_{i+1}))$
      iv. $\mathsf{sk}_{g_i} \leftarrow \mathsf{OneFSFE.KeyGen}(\mathsf{msk}_i, g_{f,\widetilde{\mathsf{st}}_i,c_i}; r_{\mathsf{KeyGen}_i})$ for $g_{f,\widetilde{\mathsf{st}}_i,c_i}$ as defined in Figure 6.
      v. If $i > 1$
         A. $c'_i \leftarrow \mathsf{Sym.Enc}(k'_i, \mathsf{sk}_{g_i})$
         B. $\mathsf{sk}'_{h_i} \leftarrow \mathsf{OneCompFE.KeyGen}(\mathsf{msk}'_i, h_{c_i,c'_i})$ for $h_{c'_i,c_i}$ as defined in Figure 7.
5. **Function Key:** Send $\mathsf{SK}_f = \mathsf{sk}_{g_1}$ to the adversary.
6. **Challenge Message:** $\mathcal{A}$ outputs a challenge message $x = x_1 \dots x_n$ for some $n \in \mathbb{N}$ and where each $x_i \in \{0,1\}^{\ell_\mathcal{X}}$.
7. **Compute p_i and ψ_i:**
   (a) $\mathsf{st}_1 = 0^{\ell_\mathcal{S}}$
   (b) For $i \in [n]$,
      i. $p_i = \widetilde{\mathsf{st}}_i \oplus \mathsf{st}_i$
      ii. $(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$
      iii. $\psi_i = \theta_i \oplus y_i$
   (c) $p_{i+1} = \widetilde{\mathsf{st}}_{i+1} \oplus \mathsf{st}_{i+1}$
8. **Challenge Ciphertext:**
   (a) For $i \in [n]$,
      i. $\mathsf{ct}_i \leftarrow \mathsf{OneFSFE.Enc}(\mathsf{msk}_i, (0^{\ell_\mathcal{X}}, 0^{\ell_\mathcal{S}}, 0^{\ell_\mathcal{S}}, 0^\lambda, 0^\lambda, 0^\lambda, 0^\lambda, 1, r_{k_i}, \psi_i))$
      ii. If $i = 1$, $\mathsf{CT}_1 = \mathsf{ct}_1$. Else, $\mathsf{CT}_i = (\mathsf{ct}_i, \mathsf{sk}'_{h_i})$
   (b) Send $\mathsf{CT} = (\mathsf{CT}_i)_{i \in [n]}$ to the adversary.
9. **Adversary's Output:** $\mathcal{A}$ outputs a bit b which is the outcome of the experiment.

Lemma 5.11. *For all adversaries $\mathcal{A}$,*

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,q,2}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{7}(1^\lambda) = 1]\right| = 0$$

where $q = q(\lambda)$ is the runtime of $\mathcal{A}$ on security parameter λ.

Proof. These hybrids are identical. Observe that if $q(\lambda)$ is the runtime of $\mathcal{A}$, then $q(\lambda) \geq n$ for any challenge message $x = x_1 \dots x_n$ output by $\mathcal{A}$ on security parameter λ. Thus, $\mathbf{Hybrid}^{\mathcal{A}}_{6,q,2}$ always uses the $\alpha_i = \alpha'_i = 1$ branches for ct_i and ct'_i just like in $\mathbf{Hybrid}^{\mathcal{A}}_{7}$. $\square$

Hybrid$_8^{\mathcal{A}}(1^\lambda)$: We write the experiment using an explicit simulator Sim. Observe that Sim is PPT.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs a function size $1^{\ell_\mathcal{F}}$, a state size $1^{\ell_\mathcal{S}}$, an input size $1^{\ell_\mathcal{X}}$, and an output size $1^{\ell_\mathcal{Y}}$. The simulator Sim receives $(1^\lambda, 1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{S}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}})$.
2. **Function Query:** $\mathcal{A}$ outputs a streaming function query $f \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$ which is sent to Sim.
3. **Compute Randomness:** Sim computes the following:
    (a) For $i \in [n+1]$,[13]
        i. $r_{\mathsf{msk}_i}, r'_{\mathsf{msk}_i}, r_{k_i}, r'_{k_i}, r'_{\mathsf{Enc}_i}, r_{\mathsf{KeyGen}_i} \leftarrow \{0,1\}^\lambda$
        ii. $\mathsf{msk}_i \leftarrow \mathsf{OneFSFE.Setup}(1^\lambda; r_{\mathsf{msk}_i})$, $\mathsf{msk}'_i \leftarrow \mathsf{OneCompFE.Setup}(1^\lambda; r'_{\mathsf{msk}_i})$, $k_i \leftarrow \mathsf{Sym.Setup}(1^\lambda; r_{k_i})$, $k'_i \leftarrow \mathsf{Sym'.Setup}(1^\lambda; r'_{k_i})$
        iii. $\widetilde{\mathsf{st}}_i \leftarrow \{0,1\}^{\ell_\mathcal{S}}$
4. **Compute** sk_{g_i} **and** sk'_{h_i}: Sim computes the following:
    (a) For $i \in [n]$,
        i. $\theta_i \leftarrow \{0,1\}^{\ell_\mathcal{Y}}$
        ii. $\mathsf{ct}'_{i+1} \leftarrow \mathsf{OneCompFE.Enc}(\mathsf{msk}'_{i+1}, (0^{\ell_\mathcal{F}}, 0^{\ell_\mathcal{S}}, 0^\lambda, 0^\lambda, 1, r'_{k_i}); r'_{\mathsf{Enc}_{i+1}})$
        iii. $c_i \leftarrow \mathsf{Sym.Enc}(k_i, (\theta_i, \mathsf{ct}'_{i+1}))$
        iv. $\mathsf{sk}_{g_i} \leftarrow \mathsf{OneFSFE.KeyGen}(\mathsf{msk}_i, g_{f,\widetilde{\mathsf{st}}_i,c_i}; r_{\mathsf{KeyGen}_i})$ for $g_{f,\widetilde{\mathsf{st}}_i,c_i}$ as defined in Figure 6.
        v. If $i > 1$
            A. $c'_i \leftarrow \mathsf{Sym.Enc}(k'_i, \mathsf{sk}_{g_i})$
            B. $\mathsf{sk}'_{h_i} \leftarrow \mathsf{OneCompFE.KeyGen}(\mathsf{msk}'_i, h_{c_i,c'_i})$ for h_{c_i,c'_i} as defined in Figure 7.
5. **Function Key:** Sim sends $\mathsf{SK}_f = \mathsf{sk}_{g_1}$ to the adversary.
6. **Challenge Message:** $\mathcal{A}$ outputs a challenge message $x = x_1 \ldots x_n$ for some $n \in \mathbb{N}$ and where each $x_i \in \{0,1\}^{\ell_\mathcal{X}}$. The simulator does *not* receive x.
7. **Challenge Message Output:** Sim receives 1^n and $y = (y_1, \ldots, y_n)$ where $y = f(x)$.
8. **Compute** ψ_i**:** Sim computes the following:
    (a) For $i \in [n]$,
        i. $\psi_i = \theta_i \oplus y_i$
9. **Challenge Ciphertext:** Sim computes the following:
    (a) For $i \in [n]$,
        i. $\mathsf{ct}_i \leftarrow \mathsf{OneFSFE.Enc}(\mathsf{msk}_i, (0^{\ell_\mathcal{X}}, 0^{\ell_\mathcal{S}}, 0^{\ell_\mathcal{S}}, 0^\lambda, 0^\lambda, 0^\lambda, 0^\lambda, 1, r_{k_i}, \psi_i))$
        ii. If $i = 1$, $\mathsf{CT}_1 = \mathsf{ct}_1$. Else, $\mathsf{CT}_i = (\mathsf{ct}_i, \mathsf{sk}'_{h_i})$
    (b) Sim sends $\mathsf{CT} = (\mathsf{ct}_i)_{i\in[n]}$ to the adversary.
10. **Adversary's Output:** $\mathcal{A}$ outputs a bit b which is the outcome of the experiment.

[13]See Remark 5.3

Lemma 5.12. *For all adversaries $\mathcal{A}$,*

$$\left|\Pr[\mathbf{Hybrid}_7^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_8^{\mathcal{A}}(1^\lambda) = 1]\right| = 0$$

Proof. $\mathbf{Hybrid}_7^{\mathcal{A}}$ and $\mathbf{Hybrid}_8^{\mathcal{A}}$ are identically distributed. Observe that as p_i is not used in the previous hybrid, we only need to compute ψ_i in step 8. However, the value of ψ_i only depends on y_i and θ_i. Thus, the simulator can compute ψ_i from y, without needing to know x. The only other change we make is that we explicitly label the challenger as a simulator. Thus, this hybrid is identically distributed to the previous one. □

Thus, our lemmas give us the following corollary:

Corollary 5.13. *If*

- PRF *and* $\mathsf{PRF2}$ *are secure PRFs,*
- Sym *and* Sym' *are secure symmetric key encryption schemes,*
- $\mathsf{OneCompFE}$ *is single-key, single-ciphertext, selective-IND-secure,*
- *and* $\mathsf{OneFSFE}$ *is single-key, single-ciphertext, function-selective-IND-secure FE*

then $\mathsf{One\text{-}sFE}$ *is single-key, single-ciphertext, function-selective-SIM-secure.*

Proof. By combining the hybrid indistinguishability lemmas above, we get that for all PPT adversaries $\mathcal{A}$,

$$\left|\Pr[\mathbf{Hybrid}_1^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_8^{\mathcal{A}}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

The corollary then follows from the fact that $\mathbf{Hybrid}_1^{\mathcal{A}}$ represents the real world security experiment $\mathsf{RealExpt}_{\mathcal{A}}^{\mathsf{One\text{-}Func\text{-}Sel\text{-}SIM}}$ and $\mathbf{Hybrid}_8^{\mathcal{A}}$ represents the idea world security experiment $\mathsf{IdealExpt}_{\mathcal{A},\mathsf{Sim}}^{\mathsf{One\text{-}Func\text{-}Sel\text{-}SIM}}$ for the PPT Sim defined in $\mathbf{Hybrid}_8^{\mathcal{A}}$. □

Corollary 5.13 then implies Theorem 5.1 as we can instantiate the required primitives from a strongly-compact, selective-IND-secure, secret-key FE scheme for $\mathsf{P/Poly}$.

6 Bootstrapping to an IND-Secure, Public-Key Streaming FE Scheme

We now construct our semi-adaptive-function-selective-IND-secure, public-key sFE scheme. We prove the following:

Theorem 6.1. *Assuming*

1. *a selective-IND-secure, public-key FE scheme for* $\mathsf{P/Poly}$
2. *a single-key, single-ciphertext, function-selective-IND-secure, secret-key, sFE scheme for* $\mathsf{P/Poly}$

there exists a semi-adaptive-function-selective-IND-secure, public-key sFE scheme for $\mathsf{P/Poly}$*.*

Remark 6.2. In fact, if the secret-key sFE scheme was adaptive-IND-secure, then our bootstrapping procedure would produce an adaptive-IND-secure, public-key sFE scheme. More precisely, assuming (1) a selective-IND-secure, public-key FE scheme for $\mathsf{P/Poly}$, and (2) a single-key, single-ciphertext, *adaptive*-IND-secure, secret-key, sFE scheme for $\mathsf{P/Poly}$, there exists an *adaptive*-IND-secure, public-key sFE scheme for $\mathsf{P/Poly}$. We do not formally prove this here, but the proof is essentially the same as that of Theorem 6.1.

Then, by applying Theorem 5.1, we get our main theorem:

Theorem 6.3. *Assuming a strongly-compact, selective-IND-secure, public-key FE scheme for* $\mathsf{P/Poly}$*, there exists a semi-adaptive-function-selective-IND-secure, public-key sFE scheme for* $\mathsf{P/Poly}$*.*

Proof. This follows from Theorem 5.1 and Theorem 6.1 since a strongly-compact, selective-IND-secure, *public-key* FE scheme for $\mathsf{P/Poly}$ implies a strongly-compact, selective-IND-secure, *secret-key* FE scheme for $\mathsf{P/Poly}$, and a single-key, single-ciphertext, function-selective-*SIM*-secure, secret-key, sFE scheme for $\mathsf{P/Poly}$ implies a single-key, single-ciphertext, function-selective-*IND*-secure, secret-key, sFE scheme for $\mathsf{P/Poly}$. □

Please refer to the technical overview (Section 2) for a high level overview of our construction. Our construction and security proof is nearly the same as in [AS16], except for a few minor modifications detailed later in Remark 6.5.

To prove Theorem 6.1, we build an sFE scheme from the following tools. As we show below, apart from $\mathsf{One\text{-}sFE}$, all of the following tools can be instantiated using a selective-IND-secure, public-key FE scheme for $\mathsf{P/Poly}$.

Tools.

- $\mathsf{One\text{-}sFE} = (\mathsf{One\text{-}sFE.Setup}, \mathsf{One\text{-}sFE.Enc}, \mathsf{One\text{-}sFE.KeyGen}, \mathsf{One\text{-}sFE.Dec})$: A single-key, single-ciphertext, function-selective-IND-secure, secret-key sFE scheme for $\mathsf{P/Poly}$.
- $\mathsf{PRF} = (\mathsf{PRF.Setup}, \mathsf{PRF.Eval})$: A secure pseudorandom function family.
- $\mathsf{PRF2} = (\mathsf{PRF2.Setup}, \mathsf{PRF2.Eval})$: A secure pseudorandom function family.
- $\mathsf{Sym} = (\mathsf{Sym.Setup}, \mathsf{Sym.Enc}, \mathsf{Sym.Dec})$: A secure symmetric key encryption scheme with pseudorandom ciphertexts.
- $\mathsf{FPFE} = (\mathsf{FPFE.Setup}, \mathsf{FPFE.Enc}, \mathsf{FPFE.KeyGen}, \mathsf{FPFE.Dec})$: A function-private-selective-IND-secure, secret-key FE scheme for $\mathsf{P/Poly}$
- $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.Enc}, \mathsf{FE.KeyGen}, \mathsf{FE.Dec})$: A selective-IND-secure, public-key FE scheme for $\mathsf{P/Poly}$.

Instantiation of the Tools. Let FE' be a selective-IND-secure, public-key FE scheme for $\mathsf{P/Poly}$.

- We can build $\mathsf{PRF}, \mathsf{PRF2}, \mathsf{Sym}$ from any one-way-function using standard cryptographic techniques (e.g. [Gol01, Gol09]). As FE' implies one-way-functions, then we can build these from FE'.
- FE' already satisfies the security requirements needed for FE.
- FE' immediately implies a selective-IND-secure, *secret-key* FE scheme SKFE' for $\mathsf{P/Poly}$. We can then build our function-private-selective-IND-secure, secret-key FE scheme FPFE for $\mathsf{P/Poly}$ by using the function-privacy transformation of [BS18] on SKFE'.

6.1 Parameters

On security parameter λ, function size $\ell_{\mathcal{F}}$, state size $\ell_{\mathcal{S}}$, input size $\ell_{\mathcal{X}}$, and output size $\ell_{\mathcal{Y}}$, we will instantiate our primitives with the following parameters:

- $\mathsf{One\text{-}sFE}$: We instantiate $\mathsf{One\text{-}sFE}$ with function size $\ell_{\mathcal{F}}$, state size $\ell_{\mathcal{S}}$, input size $\ell_{\mathcal{X}}$, and output size $\ell_{\mathcal{Y}}$. This means that we will use the following algorithms:
  - $\mathsf{One\text{-}sFE.Setup}(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}})$, $\mathsf{One\text{-}sFE.EncSetup}(1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}}, \cdot)$, $\mathsf{One\text{-}sFE.Enc}(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}}, \cdot, \cdot)$, $\mathsf{One\text{-}sFE.KeyGen}(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}}, \cdot, \cdot)$, $\mathsf{One\text{-}sFE.Dec}(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}}, \cdot, \cdot)$
- PRF: We instantiate PRF with input size λ and output size 5λ. This means that we will use the following setup algorithm: $\mathsf{PRF.Setup}(1^\lambda, 1^\lambda, 1^{5\lambda})$.
- $\mathsf{PRF2}$: We instantiate $\mathsf{PRF2}$ with input size λ and output size λ. This means that we will use the following setup algorithm: $\mathsf{PRF2.Setup}(1^\lambda, 1^\lambda, 1^\lambda)$.
- FPFE: We instantiate FPFE with
  - **Input Size:** $\ell_{\mathsf{FPFE}.m_\lambda} = \ell_{\mathsf{One\text{-}sFE.msk}_\lambda} + \ell_{\mathsf{One\text{-}sFE.Enc.st}_\lambda} + \ell_{\mathsf{PRF2}.k_\lambda} + 2$ where $\ell_{\mathsf{One\text{-}sFE.msk}_\lambda}$ is the size of master secret keys of $\mathsf{One\text{-}sFE}$, $\ell_{\mathsf{One\text{-}sFE.Enc.st}_\lambda}$ is the size of encryption states of $\mathsf{One\text{-}sFE}$, and $\ell_{\mathsf{PRF2}.k_\lambda}$ is the size of keys of $\mathsf{PRF2}$.
  - **Function Size:** ℓ_{H_λ} where ℓ_{H_λ} is the maximum of the size of H_{i,x_i,t_i} defined in Figure 8 and the size of H^*_{i,x_i,x'_i,t_i,v_i} defined in Figure 10 for any
    * $i, t_i \in \{0,1\}^\lambda$
    * $x_i, x'_i \in \{0,1\}^{\ell_{\mathcal{X}}}$
    * v_i of size $\ell_{\mathsf{One\text{-}sFE.ct}_\lambda}$ where $\ell_{\mathsf{One\text{-}sFE.ct}_\lambda}$ is the size of ciphertexts of $\mathsf{One\text{-}sFE}$

    Observe that the function size depends only on $\lambda, \ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}$ and the sizes of $\mathsf{PRF2}$, and $\mathsf{One\text{-}sFE}$.
  - **Output Size:** $\ell_{\mathsf{One\text{-}sFE.ct}_\lambda}$ where $\ell_{\mathsf{One\text{-}sFE.ct}_\lambda}$ is the size of ciphertexts of $\mathsf{One\text{-}sFE}$

  This means that we will use the following algorithms:
  - $\mathsf{FPFE.Setup}(1^\lambda, 1^{\ell_{H_\lambda}}, 1^{\ell_{\mathsf{FPFE}.m_\lambda}}, 1^{\ell_{\mathsf{One\text{-}sFE.ct}_\lambda}})$, $\mathsf{FPFE.Enc}(1^\lambda, 1^{\ell_{H_\lambda}}, 1^{\ell_{\mathsf{FPFE}.m_\lambda}}, 1^{\ell_{\mathsf{One\text{-}sFE.ct}_\lambda}}, \cdot, \cdot)$, $\mathsf{FPFE.KeyGen}(1^\lambda, 1^{\ell_{H_\lambda}}, 1^{\ell_{\mathsf{FPFE}.m_\lambda}}, 1^{\ell_{\mathsf{One\text{-}sFE.ct}_\lambda}}, \cdot, \cdot)$, $\mathsf{FPFE.Dec}(1^\lambda, 1^{\ell_{H_\lambda}}, 1^{\ell_{\mathsf{FPFE}.m_\lambda}}, 1^{\ell_{\mathsf{One\text{-}sFE.ct}_\lambda}}, \cdot, \cdot)$
- Sym: We instantiate Sym with messages of length $\ell_{\mathsf{Sym}.m_\lambda} = \ell_{\mathsf{One\text{-}sFE.sk}_\lambda} + \ell_{\mathsf{FPFE.ct}_\lambda}$ where $\ell_{\mathsf{One\text{-}sFE.sk}_\lambda}$ is the size of function keys of $\mathsf{One\text{-}sFE}$ and $\ell_{\mathsf{FPFE.ct}_\lambda}$ is the size of ciphertexts of FPFE. This means that we will use the following setup algorithm: $\mathsf{Sym.Setup}(1^\lambda, 1^{\ell_{\mathsf{Sym}.m_\lambda}})$.

- FE: We instantiate FE with
  - **Input Size:** $\ell_{\mathsf{FE}.m_\lambda} = \ell_{\mathsf{FPFE.msk}_\lambda} + \ell_{\mathsf{PRF}.k_\lambda} + 1 + \ell_{\mathsf{Sym}.k_\lambda}$ where $\ell_{\mathsf{FPFE.msk}_\lambda}$ is the size of master secret keys of FPFE, $\ell_{\mathsf{PRF}.k_\lambda}$ is the size of keys of PRF, and $\ell_{\mathsf{Sym}.k_\lambda}$ is the size of keys of Sym.
  - **Function Size:** ℓ_{G_λ} where ℓ_{G_λ} is the maximum size of $G_{f,s,c}$ defined in Figure 9 for any
    * $f \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$
    * $s \in \{0,1\}^\lambda$
    * c of size $\ell_{\mathsf{Sym.ct}_\lambda}$ where $\ell_{\mathsf{Sym.ct}_\lambda}$ is the size of ciphertexts of Sym

    Note that the function length depends only on $\lambda, \ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}$ and the sizes of PRF, $\mathsf{PRF2}$, $\mathsf{One\text{-}sFE}$, FPFE, and Sym.
  - **Output Size:** $\ell_{\mathsf{FE}.out_\lambda} = \ell_{\mathsf{One\text{-}sFE.sk}_\lambda} + \ell_{\mathsf{FPFE}.ct_\lambda}$ where $\ell_{\mathsf{One\text{-}sFE.sk}_\lambda}$ is the size of secret keys of $\mathsf{One\text{-}sFE}$ and $\ell_{\mathsf{FPFE.ct}_\lambda}$ is the size of ciphertexts of FPFE

  This means that we will use the following algorithms:

  - $\mathsf{FE.Setup}(1^\lambda, 1^{\ell_{G_\lambda}}, 1^{\ell_{\mathsf{FE}.m_\lambda}}, 1^{\ell_{\mathsf{FE}.out_\lambda}})$, $\mathsf{FE.Enc}(1^\lambda, 1^{\ell_{G_\lambda}}, 1^{\ell_{\mathsf{FE}.m_\lambda}}, 1^{\ell_{\mathsf{FE}.out_\lambda}}, \cdot, \cdot)$, $\mathsf{FE.KeyGen}(1^\lambda, 1^{\ell_{G_\lambda}}, 1^{\ell_{\mathsf{FE}.m_\lambda}}, 1^{\ell_{\mathsf{FE}.out_\lambda}}, \cdot, \cdot)$, $\mathsf{FE.Dec}(1^\lambda, 1^{\ell_{G_\lambda}}, 1^{\ell_{\mathsf{FE}.m_\lambda}}, 1^{\ell_{\mathsf{FE}.out_\lambda}}, \cdot, \cdot)$

Notation. For notational convenience, when the parameters are understood, we will often omit the security, input size, output size, message size, function size, or state size parameters from each of the algorithms listed above.

Remark 6.4. We assume without loss of generality that for security parameter λ, all algorithms only require randomness of length λ. If the original algorithm required additional randomness, we can replace it with a new algorithm that first expands the λ bits of randomness using a PRG of appropriate stretch and then runs the original algorithm. Note that this replacement does not affect the security of the above schemes (as long as $\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}$ are polynomial in λ).

6.2 Construction

We now construct our streaming FE scheme sFE. Recall that for notational convenience, we may omit the security, input size, output size, message size, function size, or state size parameters from our algorithms. For information on these parameters, please see the parameter section above.

Remark 6.5. Our construction is nearly the same as in [AS16]. Here, we are bootstrapping a single-key, single-ciphertext streaming FE scheme as opposed to a single-key, single-ciphertext FE scheme for Turing machines. There are only a few minor changes from the construction of [AS16]:

- In each function $G_{f,s,c}$, in addition to encrypting $\mathsf{One\text{-}sFE.msk}$ under FPFE, we also encrypt the starting encryption state $\mathsf{One\text{-}sFE.Enc.st}$ and a PRF key $\mathsf{PRF2}.k$. This also slightly changes the definition of each H_{i,x_i,t_i} function . In [AS16], these additional values were not needed. The proof of security can be easily modified to accommodate these values.
- For each $x = x_1 \dots x_n$, we create n FPFE function keys, one for each x_i. In [AS16], we only needed one function key. This change requires us to rely on an unbounded-key, function-private FE scheme, as opposed to the single-key, function-private FE scheme used in [AS16]. The proof of security is similar except that we perform changes across all FPFE function keys at once.
- We break the encryption algorithm of [AS16] into two parts: $\mathsf{EncSetup}$ and Enc.

We now describe our construction.

- $\mathsf{sFE.Setup}(1^\lambda, 1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{S}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}})$:
  1. $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$
  2. Output $(\mathsf{MPK} = \mathsf{FE.mpk}, \mathsf{MSK} = \mathsf{FE.msk})$.
- $\mathsf{sFE.EncSetup}(\mathsf{MPK})$:
  1. Parse $\mathsf{MPK} = \mathsf{FE.mpk}$.
  2. $\mathsf{PRF}.K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
  3. $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$
  4. $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (\mathsf{FPFE.msk}, \mathsf{PRF}.K, 0, 0^{\ell_{\mathsf{Sym}.k_\lambda}}))$.
  5. Output $\mathsf{Enc.ST} = (\mathsf{FPFE.msk}, \mathsf{FE.ct})$
- $\mathsf{sFE.Enc}(\mathsf{MPK}, \mathsf{Enc.ST}, i, x_i)$:
  1. Parse $\mathsf{Enc.ST} = (\mathsf{FPFE.msk}, \mathsf{FE.ct})$.
  2. $t_i \leftarrow \{0,1\}^\lambda$
  3. Let $H_i = H_{i,x_i,t_i}$ as defined in Figure 8.
  4. $\mathsf{FPFE.sk}_{H_i} = \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H_i)$
  5. If $i = 1$, output $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})$.
  6. Else, output $\mathsf{CT}_i = \mathsf{FPFE.sk}_{H_i}$

```
$H_{i,x_i,t_i}(\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, \mathsf{PRF2}.k, \beta)$:
  1. If $\beta = 0$
     (a) $r_i \leftarrow \mathsf{PRF2.Eval}(\mathsf{PRF2}.k, t_i)$
     (b) Output $\mathsf{One\text{-}sFE.ct}_i \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, i, x_i; r_i)$
  2. Else, output $\perp$
```

Figure 8

- $\mathsf{sFE.KeyGen}(\mathsf{MSK}, f)$:
  1. Parse $\mathsf{MSK} = \mathsf{FE.msk}$.
  2. $s \leftarrow \{0,1\}^{\lambda}$
  3. $c \leftarrow \{0,1\}^{\ell_{\mathsf{Sym.ct}_\lambda}}$
  4. Let $G = G_{f,s,c}$ as defined in Figure 9.
  5. $\mathsf{FE.sk}_G \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G)$
  6. Output $\mathsf{SK}_f = \mathsf{FE.sk}_G$

```
$G_{f,s,c}(\mathsf{FPFE.msk}, \mathsf{PRF}.K, \alpha, \mathsf{Sym}.k)$:
  1. If $\alpha = 0$
     (a) $(r_{\mathsf{Setup}}, r_{\mathsf{KeyGen}}, r_{\mathsf{EncSetup}}, r_{\mathsf{PRF2}}, r_{\mathsf{Enc}}) \leftarrow \mathsf{PRF.Eval}(\mathsf{PRF}.K, s)$
     (b) $\mathsf{One\text{-}sFE.msk} \leftarrow \mathsf{One\text{-}sFE.Setup}(1^{\lambda}; r_{\mathsf{Setup}})$
     (c) $\mathsf{One\text{-}sFE.Enc.st} \leftarrow \mathsf{One\text{-}sFE.EncSetup}(\mathsf{One\text{-}sFE.msk}; r_{\mathsf{EncSetup}})$
     (d) $\mathsf{One\text{-}sFE.sk}_f \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}, f; r_{\mathsf{KeyGen}})$
     (e) $\mathsf{PRF2}.k \leftarrow \mathsf{PRF2.Setup}(1^{\lambda}; r_{\mathsf{PRF2}})$
     (f) $\mathsf{FPFE.ct} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, \mathsf{PRF2}.k, 0); r_{\mathsf{Enc}})$
     (g) Output $(\mathsf{One\text{-}sFE.sk}_f, \mathsf{FPFE.ct})$
  2. Else
     (a) Output $(\mathsf{One\text{-}sFE.sk}_f, \mathsf{FPFE.ct}) \leftarrow \mathsf{Sym.Dec}(\mathsf{Sym}.k, c)$
```

Figure 9

- $\mathsf{sFE.Dec}(\mathsf{SK}_f, \mathsf{Dec.ST}_i, i, \mathsf{CT}_i)$:
  1. If $i = 1$
     (a) Parse $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})$ and $\mathsf{SK}_f = \mathsf{FE.sk}_G$.
     (b) $(\mathsf{One\text{-}sFE.sk}_f, \mathsf{FPFE.ct}) = \mathsf{FE.Dec}(\mathsf{FE.sk}_G, \mathsf{FE.ct})$
     (c) Set $\mathsf{One\text{-}sFE.Dec.st}_1 = \perp$.
  2. If $i > 1$
     (a) Parse $\mathsf{CT}_i = \mathsf{FPFE.sk}_{H_i}$
     (b) Parse $\mathsf{Dec.ST}_i = (\mathsf{One\text{-}sFE.sk}_f, \mathsf{FPFE.ct}, \mathsf{One\text{-}sFE.Dec.st}_i)$

3. $\mathsf{One\text{-}sFE.ct}_i = \mathsf{FPFE.Dec}(\mathsf{FPFE.sk}_{H_i}, \mathsf{FPFE.ct})$
4. $(y_i, \mathsf{One\text{-}sFE.Dec.st}_{i+1}) = \mathsf{One\text{-}sFE.Dec}(\mathsf{One\text{-}sFE.sk}_f, \mathsf{One\text{-}sFE.Dec.st}_i, i, \mathsf{One\text{-}sFE.ct}_i)$
5. Output $(y_i, \mathsf{Dec.ST}_{i+1} = (\mathsf{One\text{-}sFE.sk}_f, \mathsf{FPFE.ct}, \mathsf{One\text{-}sFE.Dec.st}_{i+1}))$

We also define the following function which will be used in our security proof.

```
$H^*_{i,x_i,x'_i,t_i,v_i}(\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, \mathsf{PRF2}.k, \beta)$:

  • If $\beta = 0$
      1. $r_i \leftarrow \mathsf{PRF2.Eval}(\mathsf{PRF2}.k, t_i)$
      2. Output $\mathsf{One\text{-}sFE.ct}_i \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, i, x_i; r_i)$
  • If $\beta = 1$
      1. $r_i \leftarrow \mathsf{PRF2.Eval}(\mathsf{PRF2}.k, t_i)$
      2. Output $\mathsf{One\text{-}sFE.ct}_i \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, i, x'_i; r_i)$
  • Else, output $v_i$
```

Figure 10

6.3 Correctness and Efficiency

Efficiency: Using our discussion above on parameters, it is easy to see that the size and runtime of all algorithms of One-sFE on security parameter λ, function size $\ell_{\mathcal{F}}$, state size $\ell_{\mathcal{S}}$, input size $\ell_{\mathcal{X}}$, and output size $\ell_{\mathcal{Y}}$ are $\mathsf{poly}(\lambda, \ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}})$.

Correctness Intuition: Our ciphertext consists of $(\mathsf{FE.ct}, \{\mathsf{FPFE.sk}_{H_i}\}_{i\in[n]})$, and our function key consists of $\mathsf{SK}_f = \mathsf{FE.sk}_G$. We can combine $\mathsf{FE.ct}$ and $\mathsf{FE.sk}_G$ via FE decryption to get a function key $\mathsf{One\text{-}sFE.sk}_f$ for f under $\mathsf{One\text{-}sFE.msk}$, and a ciphertext $\mathsf{FPFE.ct}$ containing $\mathsf{One\text{-}sFE.msk}$. Then, for $i \in [n]$, we can combine $\mathsf{FPFE.ct}$ and $\mathsf{FPFE.sk}_{H_i}$ to get the i^{th} ciphertext $\mathsf{One\text{-}sFE.ct}_i$ of the encryption of x under $\mathsf{One\text{-}sFE.msk}$. We can then combine $\mathsf{One\text{-}sFE.sk}_f$ and $\{\mathsf{One\text{-}sFE.ct}_i\}_{i\in[n]}$ using One-sFE decryption to compute $f(x)$.

Correctness: More formally, let p be any polynomial and consider any λ and any $\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}} \leq p(\lambda)$. Let SK_f be a function key for function $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$, and let $\mathsf{CT} = \{\mathsf{CT}_i\}_{i\in[n]}$ be a ciphertext for x where $x = x_1 \dots x_n$ for some $n \in [2^\lambda]$ and where each $x_i \in \{0,1\}^{\ell_{\mathcal{X}}}$.

First parse $\mathsf{SK}_f = \mathsf{FE.sk}_G$, $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})$, and $\mathsf{CT}_i = \mathsf{FPFE.sk}_{H_i}$ for $i \in [n]\backslash\{1\}$. Then, by correctness of FE, except with negligible probability,

$$\begin{aligned}\mathsf{FE.Dec}(\mathsf{FE.sk}_G, \mathsf{FE.ct}) &= G_{f,s,c}(\mathsf{FPFE.msk}, \mathsf{PRF}.K, 0, 0^{\ell_{\mathsf{Sym}.k_\lambda}}) \\ &= (\mathsf{One\text{-}sFE.sk}_f, \mathsf{FPFE.ct})\end{aligned}$$

where $\mathsf{One\text{-}sFE.sk}_f$ is a One-sFE function key for f generated under $\mathsf{One\text{-}sFE.msk}$, and $\mathsf{FPFE.ct}$ is an FPFE ciphertext encrypting $(\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, \mathsf{PRF2}.k, 0)$ as defined by

$$\begin{aligned}
&(r_{\mathsf{Setup}}, r_{\mathsf{KeyGen}}, r_{\mathsf{EncSetup}}, r_{\mathsf{PRF2}}, r_{\mathsf{Enc}}) \leftarrow \mathsf{PRF.Eval}(\mathsf{PRF}.K, s) \\
&\mathsf{One\text{-}sFE.msk} \leftarrow \mathsf{One\text{-}sFE.Setup}(1^\lambda; r_{\mathsf{Setup}}) \\
&\mathsf{One\text{-}sFE.Enc.st} \leftarrow \mathsf{One\text{-}sFE.EncSetup}(\mathsf{One\text{-}sFE.msk}) \\
&\mathsf{One\text{-}sFE.sk}_f \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}, f; r_{\mathsf{KeyGen}}) \\
&\mathsf{PRF2}.k \leftarrow \mathsf{PRF2.Setup}(1^\lambda; r_{\mathsf{PRF2}}) \\
&\mathsf{FPFE.ct} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, \mathsf{PRF2}.k, 0); r_{\mathsf{Enc}})
\end{aligned}$$

Then, by correctness of FPFE, except with negligible probability, for all $i \in [n]$,

$$\begin{aligned}
\mathsf{FPFE.Dec}(\mathsf{FPFE.sk}_{H_i}, \mathsf{FPFE.ct}) &= H_{i,x_i,t_i}(\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, \mathsf{PRF2}.k, 0) \\
&= \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, i, x_i; \mathsf{PRF2.Eval}(\mathsf{PRF2}.k, t_i)) \\
&= \mathsf{One\text{-}sFE.ct}_i
\end{aligned}$$

where $\mathsf{One\text{-}sFE.ct}_i$ is the i^{th} $\mathsf{One\text{-}sFE}$ ciphertext for x under $\mathsf{One\text{-}sFE.msk}$. Thus, if $\mathsf{One\text{-}sFE.Dec.st}_1 = \perp$ is the proper starting decryption state for $\mathsf{One\text{-}sFE}$, and if we define $\mathsf{One\text{-}sFE.Dec.st}_i$ for $i > 1$ inductively by

$$(y_i, \mathsf{One\text{-}sFE.Dec.st}_{i+1}) = \mathsf{One\text{-}sFE.Dec}(\mathsf{One\text{-}sFE.sk}_f, \mathsf{One\text{-}sFE.Dec.st}_i, i, \mathsf{One\text{-}sFE.ct}_i)$$

then by correctness of $\mathsf{One\text{-}sFE}$, except with negligible probability, $y = y_1 \dots y_n = f(x)$. Thus, for $i = 1$ and using the values we defined above,

$$\begin{aligned}
\mathsf{sFE.Dec}(\mathsf{SK}_f, \mathsf{Dec.ST}_1, 1, \mathsf{CT}_1) &= \mathsf{sFE.Dec}(\mathsf{FE.sk}_G, \perp, 1, (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})) \\
&= (y_1, \mathsf{Dec.ST}_2 = (\mathsf{One\text{-}sFE.sk}_f, \mathsf{FPFE.ct}, \mathsf{One\text{-}sFE.Dec.st}_2))
\end{aligned}$$

Therefore, for $i > 1$, using the values defined above,

$$\begin{aligned}
\mathsf{sFE.Dec}(\mathsf{SK}_f, \mathsf{Dec.ST}_i, i, \mathsf{CT}_i) &= \mathsf{sFE.Dec}(\mathsf{FE.sk}_G, (\mathsf{One\text{-}sFE.sk}_f, \mathsf{FPFE.ct}, \mathsf{One\text{-}sFE.Dec.st}_i), i, \mathsf{FPFE.sk}_{H_i}) \\
&= (y_i, \mathsf{Dec.ST}_{i+1} = (\mathsf{One\text{-}sFE.sk}_f, \mathsf{FPFE.ct}, \mathsf{One\text{-}sFE.Dec.st}_{i+1}))
\end{aligned}$$

Therefore, decryption correctly outputs $y = f(x)$.

6.4 Security

As the security proof is very similar to the one in [AS16], we defer it to Appendix D.

7 Acknowledgements

This research was supported in part from a Simons Investigator Award, DARPA SIEVE award, NTT Research, NSF Frontier Award 1413955, BSF grant 2012378, a Xerox Faculty Research Award, a Google Faculty Research Award, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through Award HR00112020024. The views expressed are those of the authors and do not reflect the official policy or position of any funding source or the U.S. Government.

8 References

[AAB15] Benny Applebaum, Jonathan Avron, and Christina Brzuska. Arithmetic cryptography: Extended abstract. In Tim Roughgarden, editor, *ITCS 2015*, pages 143–151. ACM, January 2015.

[ABDP15] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple functional encryption schemes for inner products. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 733–751. Springer, Heidelberg, March / April 2015.

[ABSV15] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive security in functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 657–677. Springer, Heidelberg, August 2015.

[Agr19] Shweta Agrawal. Indistinguishability obfuscation without multilinear maps: New methods for bootstrapping and instantiation. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 191–225. Springer, Heidelberg, May 2019.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 308–326. Springer, Heidelberg, August 2015.

[AJL$^+$19] Prabhanjan Ananth, Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. Indistinguishability obfuscation without multilinear maps: New paradigms via low degree weak pseudorandomness and security amplification. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 284–332. Springer, Heidelberg, August 2019.

[AJS15] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation from functional encryption for simple functions. Cryptology ePrint Archive, Paper 2015/730, 2015. `https://eprint.iacr.org/2015/730`.

[AJS18] Prabhanjan Ananth, Aayush Jain, and Amit Sahai. Indistinguishability obfuscation without multilinear maps: io from lwe, bilinear maps, and weak pseudorandomness. Cryptology ePrint Archive, Paper 2018/615, 2018. `https://eprint.iacr.org/2018/615`.

[ALS16] Shweta Agrawal, Benoît Libert, and Damien Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 333–362. Springer, Heidelberg, August 2016.

[AS16] Prabhanjan Vijendra Ananth and Amit Sahai. Functional encryption for turing machines. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 125–153. Springer, Heidelberg, January 2016.

[AS17] Prabhanjan Ananth and Amit Sahai. Projective arithmetic functional encryption and indistinguishability obfuscation from degree-5 multilinear maps. In Jean-Sébastien

Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 152–181. Springer, Heidelberg, April / May 2017.

[BCFG17] Carmen Elisabetta Zaira Baltico, Dario Catalano, Dario Fiore, and Romain Gay. Practical functional encryption for quadratic functions with applications to predicate encryption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 67–98. Springer, Heidelberg, August 2017.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 896–912. ACM Press, October 2018.

[BFKL94] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 278–291. Springer, Heidelberg, August 1994.

[BGG$^+$14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556. Springer, Heidelberg, May 2014.

[BGJS17] Saikrishna Badrinarayanan, Vipul Goyal, Aayush Jain, and Amit Sahai. A note on vrfs from verifiable functional encryption. *IACR Cryptology ePrint Archive*, 2017:51, 2017.

[Bit17] Nir Bitansky. Verifiable random functions from non-interactive witness-indistinguishable proofs. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 567–594. Springer, Heidelberg, November 2017.

[BS18] Zvika Brakerski and Gil Segev. Function-private functional encryption in the private-key setting. *Journal of Cryptology*, 31(1):202–225, January 2018.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Heidelberg, March 2011.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In Venkatesan Guruswami, editor, *56th FOCS*, pages 171–190. IEEE Computer Society Press, October 2015.

[GGG$^+$14] Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 578–602. Springer, Heidelberg, May 2014.

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

[GGHZ16] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Functional encryption without obfuscation. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part II*, volume 9563 of *LNCS*, pages 480–511. Springer, Heidelberg, January 2016.

[GHKW17] Rishab Goyal, Susan Hohenberger, Venkata Koppula, and Brent Waters. A generic approach to constructing and proving verifiable random functions. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 537–566. Springer, Heidelberg, November 2017.

[GKP$^+$13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 555–564. ACM Press, June 2013.

[Gol01] Oded Goldreich. *Foundations of Cryptography, Volume 1, Basic Tools*, volume 1. Cambridge university press, 2001.

[Gol09] Oded Goldreich. *Foundations of Cryptography, Volume 2, Basic Applications*, volume 2. Cambridge university press, 2009.

[GPS16] Sanjam Garg, Omkant Pandey, and Akshayaram Srinivasan. Revisiting the cryptographic hardness of finding a nash equilibrium. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 579–604. Springer, Heidelberg, August 2016.

[GPSZ17] Sanjam Garg, Omkant Pandey, Akshayaram Srinivasan, and Mark Zhandry. Breaking the sub-exponential barrier in obfustopia. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 156–181. Springer, Heidelberg, April / May 2017.

[GS16] Sanjam Garg and Akshayaram Srinivasan. Single-key to multi-key functional encryption with polynomial loss. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 419–442. Springer, Heidelberg, October / November 2016.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 162–179. Springer, Heidelberg, August 2012.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 503–523. Springer, Heidelberg, August 2015.

[HJO$^+$16] Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. Adaptively secure garbled circuits from one-way functions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 149–178. Springer, Heidelberg, August 2016.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, Heidelberg, August 2008.

[JLMS19] Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. How to leverage hardness of constant-degree expanding polynomials overa $\mathbb{R}$ to build $i\mathcal{O}$. In Yuval Ishai and

Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 251–281. Springer, Heidelberg, May 2019.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, page 60–73, ACM, 2021.

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in NC^0. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 670–699. Springer, Heidelberg, May / June 2022.

[Lin16] Huijia Lin. Indistinguishability obfuscation from constant-degree graded encoding schemes. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 28–57. Springer, Heidelberg, May 2016.

[Lin17] Huijia Lin. Indistinguishability obfuscation from SXDH on 5-linear maps and locality-5 PRGs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 599–629. Springer, Heidelberg, August 2017.

[LM16] Baiyu Li and Daniele Micciancio. Compactness vs collusion resistance in functional encryption. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 443–468. Springer, Heidelberg, October / November 2016.

[LT17] Huijia Lin and Stefano Tessaro. Indistinguishability obfuscation from trilinear maps and block-wise local PRGs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 630–660. Springer, Heidelberg, August 2017.

[LV16] Huijia Lin and Vinod Vaikuntanathan. Indistinguishability obfuscation from DDH-like assumptions on constant-degree graded encodings. In Irit Dinur, editor, *57th FOCS*, pages 11–20. IEEE Computer Society Press, October 2016.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. *IACR Cryptology ePrint Archive*, 2010:556, 2010.

[SW05] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473. Springer, Heidelberg, May 2005.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.

A [JLS22] Assumptions

In this section, we detail the assumptions used in [JLS22] to build their sublinear, single-key, selective-IND-secure, public-key FE scheme for $\mathsf{P/Poly}$.

Definition A.1 (δ-LPN Assumption [BFKL94, IPS08, AAB15, BCGI18]). *Let $\delta \in (0,1)$. We say that the δ-LPN Assumption is true if the following holds: For any constant $\eta_p > 0$, any function $p : \mathbb{N} \to \mathbb{N}$ such that for every $\ell \in \mathbb{N}$, $p(\ell)$ is a prime of ℓ^{η_p} bits, any constant $\eta_n > 0$, we set $p = p(\ell), n = n(\ell) = \ell^{\eta_n}$, and $r = r(\ell) = \ell^{-\delta}$, and we require that the following two distributions are computationally indistinguishable:*

$$\left\{(\boldsymbol{A}, \boldsymbol{b} = \boldsymbol{s} \cdot \boldsymbol{A} + \boldsymbol{e}) \mid \boldsymbol{A} \leftarrow \mathbb{Z}_p^{\ell \times n}, \boldsymbol{s} \leftarrow \mathbb{Z}_p^{1 \times n}, \boldsymbol{e} \leftarrow \mathcal{D}_r^{1 \times n}(p)\right\}_{\ell \in \mathbb{N}}$$

$$\left\{(\boldsymbol{A}, \boldsymbol{u}) \mid \boldsymbol{A} \leftarrow \mathbb{Z}_p^{\ell \times n}, \boldsymbol{u} \leftarrow \mathbb{Z}_p^{1 \times n}\right\}_{\ell \in \mathbb{N}}$$

where $e \leftarrow \mathcal{D}_r(p)$ is a generalized Bernoulli distribution, i.e. e is sampled randomly from $\mathbb{Z}_p$ with probability $r = \ell^{-\delta}$ and set to be 0 with probability $1 - r$.

Definition A.2 (Pseudorandom Generator). *A stretch-$m(\cdot)$ pseudorandom generator is a Boolean function $\mathsf{PRG} : \{0,1\}^* \to \{0,1\}^*$ mapping n-bit inputs to $m(n)$-bit outputs (also known as the stretch) that is computable by a uniform PPT machine, and for any non-uniform PPT adversary $\mathcal{A}$, there exists a negligible function μ such that for all $n \in \mathbb{N}$,*

$$\left| \Pr_{r \leftarrow \{0,1\}^n}[\mathcal{A}(\mathsf{PRG}(r)) = 1] - \Pr_{z \leftarrow \{0,1\}^m}[\mathcal{A}(z) = 1] \right| < \mu(n)$$

Further, a PRG is said to be in NC^0 if PRG is implementable by a uniformly efficiently generatable NC^0 circuit.

Definition A.3 (DLIN Assumption). *The decision linear (DLIN) assumption over prime order symmetric bilinear groups is stated as follows: Given an appropriate prime p, two groups $\mathcal{G}, \mathcal{G}_T$ are chosen of order p such that there exists an efficiently computable nontrivial bilinear map $e : \mathcal{G} \times \mathcal{G} \to \mathcal{G}_T$. Canonical generators g for $\mathcal{G}$ and g_T for $\mathcal{G}_T$ are also computed. Then, the DLIN assumption requires that the following computational indistinguishability holds:*

$$\{(g^x, g^y, g^z, g^{xa}, g^{yb}, g^{z(a+b)}) \mid x, y, z, a, b \leftarrow \mathbb{Z}_p\} \approx_c \{(g^x, g^y, g^z, g^{xa}, g^{yb}, g^{zc}) \mid x, y, z, a, b, c \leftarrow \mathbb{Z}_p\}$$

B Preliminaries Continued

B.1 Standard Notions

Definition B.1 (PRF). *A pseudorandom function family (PRF) with key space $\mathcal{K} = \{\mathcal{K}_{\lambda,n,m}\}_{\lambda,n,m \in \mathbb{N}}$ is a tuple of PPT algorithms $\mathsf{PRF} = (\mathsf{PRF.Setup}, \mathsf{PRF.Eval})$ where*

- $\mathsf{PRF.Setup}(1^\lambda, 1^n, 1^m)$: *takes as input the security parameter λ, an input length n, and an output length m, and outputs a key $k \in \mathcal{K}_{\lambda,n,m}$*
- $\mathsf{PRF.Eval}(k, x)$ *takes as input a key $k \in \mathcal{K}_{\lambda,n,m}$ and an input $x \in \{0,1\}^n$, and outputs a value $y \in \{0,1\}^m$.*

Security requires that there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$ and every PPT adversary $\mathcal{A}$,

$$\left| \Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{PRF}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{PRF}}(1^\lambda, 1) = 1] \right| \leq \mu(\lambda)$$

where for each $b \in \{0,1\}$ and $\lambda \in \mathbb{N}$, we define

$\mathsf{Expt}_{\mathcal{A}}^{\mathsf{PRF}}(1^\lambda, b)$

1. ***Parameters:*** *$\mathcal{A}$ takes as input 1^λ and outputs an input size 1^n and an output size 1^m*
2. ***Setup:***
   (a) *If $b = 0$, sample $\mathsf{PRF}.k \leftarrow \mathsf{PRF.Setup}(1^\lambda, 1^n, 1^m)$.*
   (b) *If $b = 1$, sample $R \leftarrow \mathcal{R}_{n,m}$ where $\mathcal{R}_{n,m}$ is the set of all functions from $\{0,1\}^n$ to $\{0,1\}^m$.*
3. ***PRF Queries:*** *The following can be repeated a polynomial number of times:*
   (a) *$\mathcal{A}$ outputs an input $x \in \{0,1\}^n$*
   (b) *If $b = 0$, send $y = \mathsf{PRF.Eval}(\mathsf{PRF}.k, x)$ to $\mathcal{A}$*
   (c) *If $b = 1$, send $y = R(x)$ to $\mathcal{A}$*
4. ***Experiment Outcome:*** *$\mathcal{A}$ outputs a bit b' which is the output of the experiment.*

Definition B.2 (Symmetric Key Encryption)**.** *A symmetric key encryption scheme with key space $\mathcal{K} = \{\mathcal{K}_{\lambda,n}\}_{\lambda,n\in\mathbb{N}}$ and ciphertext size $m(\cdot)$ is a tuple of PPT algorithms $\mathsf{Sym} = (\mathsf{Sym.Setup}, \mathsf{Sym.Enc}, \mathsf{Sym.Dec})$ where*

- $\mathsf{Sym.Setup}(1^\lambda, 1^n)$*: takes as input the security parameter λ and an input length n and outputs a secret key $k \in \mathcal{K}_{\lambda,n}$*
- $\mathsf{Sym.Enc}(k, x)$*: takes as input a secret key $k \in \mathcal{K}_{\lambda,n}$ and a message $x \in \{0,1\}^n$ and outputs an encryption $\mathsf{ct} \in \{0,1\}^{m(\lambda,n)}$ of x.*
- $\mathsf{Sym.Dec}(k, \mathsf{ct})$*: takes as input a secret key $k \in \mathcal{K}_{\lambda,n}$ and a ciphertext $\mathsf{ct} \in \{0,1\}^{m(\lambda,n)}$ and outputs a value $y \in \{0,1\}^n$.*

Correctness requires that for all polynomials p, there exists a negligible function η such that for all $\lambda \in \mathbb{N}$, all $n \leq p(\lambda)$, and every $x \in \{0,1\}^n$,

$$\Pr\left[\mathsf{Sym.Dec}(k, \mathsf{Sym.Enc}(k, x)) = x : k \leftarrow \mathsf{Sym.Setup}(1^\lambda, 1^n)\right] \geq 1 - \eta(\lambda)$$

Security requires that there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$ and every PPT adversary $\mathcal{A}$,

$$\left|\Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{Sym}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{Sym}}(1^\lambda, 1) = 1]\right| \leq \mu(\lambda)$$

where for each $b \in \{0,1\}$ and $\lambda \in \mathbb{N}$, we define

$\mathsf{Expt}_{\mathcal{A}}^{\mathsf{Sym}}(1^\lambda, b)$

1. ***Parameters:*** *$\mathcal{A}$ takes as input 1^λ and outputs an input size 1^n.*
2. ***Setup:*** *$k \leftarrow \mathsf{Sym.Setup}(1^\lambda, 1^n)$*
3. ***Challenge Message Queries:*** *The following can be repeated any polynomial number of times:*
   (a) *$\mathcal{A}$ outputs a challenge message pair (x_0, x_1) where $x_0, x_1 \in \{0,1\}^n$.*

(b) $\mathsf{ct}_b \leftarrow \mathsf{Sym.Enc}(k, x_b)$

(c) *Sent* ct_b *to* $\mathcal{A}$.

4. ***Experiment Outcome:*** *$\mathcal{A}$ outputs a bit b' which is the output of the experiment.*

We will sometimes require that our symmetric key encryption scheme has pseudorandom ciphertexts. Intuitively, this means that ciphertexts should be indistinguishable from random strings of the same size.

Definition B.3 (Symmetric Key Encryption with Pseudorandom Ciphertexts)**.** *A symmetric key encryption scheme with key space $\mathcal{K} = \{\mathcal{K}_{\lambda,n}\}_{\lambda,n\in\mathbb{N}}$ and ciphertext size $m(\cdot)$ has pseudorandom ciphertexts if there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$ and every PPT adversary $\mathcal{A}$,*

$$\left|\Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{Sym\text{-}Pseudorandom\text{-}CT}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{Sym\text{-}Pseudorandom\text{-}CT}}(1^\lambda, 1) = 1]\right| \leq \mu(\lambda)$$

where for each $b \in \{0,1\}$ and $\lambda \in \mathbb{N}$, we define

$\mathsf{Expt}_{\mathcal{A}}^{\mathsf{Sym\text{-}Pseudorandom\text{-}CT}}(1^\lambda, b)$

1. ***Parameters:*** *$\mathcal{A}$ takes as input 1^λ and outputs an input size 1^n.*
2. ***Setup:*** $k \leftarrow \mathsf{Sym.Setup}(1^\lambda, 1^n)$
3. ***Challenge Message Queries:*** *The following can be repeated any polynomial number of times:*

   (a) *$\mathcal{A}$ outputs a challenge message x where $x \in \{0,1\}^n$.*

   (b) *If $b = 0$,* $\mathsf{ct} \leftarrow \mathsf{Sym.Enc}(k, x)$.

   (c) *If $b = 1$,* $\mathsf{ct} \leftarrow \{0,1\}^{m(\lambda,n)}$

   (d) *Send* ct *to* $\mathcal{A}$.
4. ***Experiment Outcome:*** *$\mathcal{A}$ outputs a bit b' which is the output of the experiment.*

B.2 Secret-Key Functional Encryption

In this section, we formally define secret-key functional encryption.

Definition B.4 (Secret-Key Functional Encryption)**.** *A secret-key functional encryption scheme for* $\mathsf{P/Poly}$ *is a tuple of PPT algorithms* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *defined as follows:*[14]

- $\mathsf{Setup}(1^\lambda, 1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}})$*: takes as input the security parameter λ, a function size $\ell_\mathcal{F}$, an input size $\ell_\mathcal{X}$, and an output size $\ell_\mathcal{Y}$, and outputs the master secret key* msk.
- $\mathsf{Enc}(\mathsf{msk}, x)$*: takes as input the master secret key* msk *and a message $x \in \{0,1\}^{\ell_\mathcal{X}}$, and outputs an encryption* ct *of x.*
- $\mathsf{KeyGen}(\mathsf{msk}, f)$*: takes as input the master secret key* msk *and a function $f \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$, and outputs a function key* sk_f.

[14]We also allow $\mathsf{Enc}, \mathsf{KeyGen}$, and Dec to additionally receive parameters $1^\lambda, 1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}}$ as input, but omit them from our notation for convenience.

- $\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct})$*: takes as input a function key* sk_f *and a ciphertext* ct*, and outputs a value* $y \in \{0,1\}^{\ell_{\mathcal{Y}}}$*.*

FE *satisfies **correctness** if for all polynomials* p*, there exists a negligible function* μ *such that for all* $\lambda \in \mathbb{N}$*, all* $\ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}} \le p(\lambda)$*, all* $x \in \{0,1\}^{\ell_{\mathcal{X}}}$*, and all* $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$*,*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}_x) = f(x) : \begin{array}{c} \mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}}) \\ \mathsf{ct}_x \leftarrow \mathsf{Enc}(\mathsf{msk}, x) \\ \mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f) \end{array}\right] \ge 1 - \mu(\lambda).$$

Selective-IND-security requires the challenge messages to be sent before the function queries.

Definition B.5 (Selective-IND Security)**.** *A secret-key functional encryption scheme* FE *for* $\mathsf{P/Poly}$ *is selective-IND-secure if there exists a negligible function* μ *such that for all* $\lambda \in \mathbb{N}$ *and every PPT adversary* $\mathcal{A}$*,*

$$\left|\Pr[\mathsf{SKExpt}_{\mathcal{A}}^{\mathsf{Sel\text{-}IND}}(1^\lambda, 0) = 1] - \Pr[\mathsf{SKExpt}_{\mathcal{A}}^{\mathsf{Sel\text{-}IND}}(1^\lambda, 1) = 1]\right| \le \mu(\lambda)$$

where for each $b \in \{0,1\}$ *and* $\lambda \in \mathbb{N}$*, we define*

$\mathsf{SKExpt}_{\mathcal{A}}^{\mathsf{Sel\text{-}IND}}(1^\lambda, b)$

1. ***Parameters**:* $\mathcal{A}$ *takes as input* 1^λ*, and outputs a function size* $1^{\ell_{\mathcal{F}}}$*, an input size* $1^{\ell_{\mathcal{X}}}$*, and an output size* $1^{\ell_{\mathcal{Y}}}$*.*
2. ***Challenge Messages**:* $\mathcal{A}$ *outputs challenge message pairs* $\{(x_{0,i}, x_{1,i})\}_{i\in[T]}$ *for some* T *chosen by the adversary where* $x_{0,i}, x_{1,i} \in \{0,1\}^{\ell_{\mathcal{X}}}$ *for all* $i \in [T]$*.*
3. ***Setup and Challenge Ciphertexts**:*
   (a) $\mathsf{msk} \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}})$
   (b) *For* $i \in [T]$*, compute* $\mathsf{ct}_i \leftarrow \mathsf{FE.Enc}(\mathsf{msk}, x_{b,i})$ *and send* ct_i *to* $\mathcal{A}$*.*
4. ***Function Queries**: The following can be repeated any polynomial number of times:*
   (a) $\mathcal{A}$ *outputs a function query* $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$
   (b) $\mathsf{sk}_f \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}, f)$
   (c) *Send* sk_f *to* $\mathcal{A}$
5. ***Experiment Outcome**:* $\mathcal{A}$ *outputs a bit* b'*. The output of the experiment is set to* 1 *if* $b = b'$ *and* $f(x_{0,i}) = f(x_{1,i})$ *for all functions* f *queried by the adversary and all* $i \in [T]$*.*

Function-selective-IND-security requires the function queries to be sent before the challenge message queries.

Definition B.6 (Function-Selective-IND-Security)**.** *A secret-key functional encryption scheme* FE *for* $\mathsf{P/Poly}$ *is function-selective-IND-secure if there exists a negligible function* μ *such that for all* $\lambda \in \mathbb{N}$ *and every PPT adversary* $\mathcal{A}$*,*

$$\left|\Pr[\mathsf{SKExpt}_{\mathcal{A}}^{\mathsf{Func\text{-}Sel\text{-}IND}}(1^\lambda, 0) = 1] - \Pr[\mathsf{SKExpt}_{\mathcal{A}}^{\mathsf{Func\text{-}Sel\text{-}IND}}(1^\lambda, 1) = 1]\right| \le \mu(\lambda)$$

where for each $b \in \{0,1\}$ *and* $\lambda \in \mathbb{N}$*, we define*

$\mathsf{SKExpt}_{\mathcal{A}}^{\mathsf{Func\text{-}Sel\text{-}IND}}(1^\lambda, b)$

1. ***Parameters:*** *$\mathcal{A}$ takes as input 1^λ, and outputs a function size $1^{\ell_\mathcal{F}}$, an input size $1^{\ell_\mathcal{X}}$, and an output size $1^{\ell_\mathcal{Y}}$.*
2. ***Setup:*** $\mathsf{msk} \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}})$
3. ***Function Queries:*** *The following can be repeated any polynomial number of times:*
   (a) *$\mathcal{A}$ outputs a function query $f \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$*
   (b) $\mathsf{sk}_f \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}, f)$
   (c) *Send sk_f to $\mathcal{A}$*
4. ***Challenge Messages:*** *$\mathcal{A}$ outputs challenge message pairs $\{(x_{0,i}, x_{1,i})\}_{i\in[T]}$ for some T chosen by the adversary where $x_{0,i}, x_{1,i} \in \{0,1\}^{\ell_\mathcal{X}}$ for all $i \in [T]$.*
5. ***Challenge Ciphertexts:*** *For $i \in [T]$, compute $\mathsf{ct}_i \leftarrow \mathsf{FE.Enc}(\mathsf{msk}, x_{b,i})$ and send ct_i to $\mathcal{A}$.*
6. ***Experiment Outcome:*** *$\mathcal{A}$ outputs a bit b'. The output of the experiment is set to 1 if $b = b'$ and $f(x_{0,i}) = f(x_{1,i})$ for all functions f queried by the adversary and all $i \in [T]$.*

C Additional Streaming FE Definitions

C.1 Secret-Key Streaming FE

In this section, we define additional notions of streaming FE. First, we define secret-key streaming FE.

Definition C.1 (Secret-Key Streaming FE)**.** *A secret-key streaming functional encryption scheme for $\mathsf{P/Poly}$ is a tuple of PPT algorithms $\mathsf{sFE} = (\mathsf{Setup}, \mathsf{EncSetup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ defined as follows:*[15]

1. $\mathsf{Setup}(1^\lambda, 1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{S}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}})$*: takes as input the security parameter λ, a function size $\ell_\mathcal{F}$, a state size $\ell_\mathcal{S}$, an input size $\ell_\mathcal{X}$, and an output size $\ell_\mathcal{Y}$, and outputs the master secret key* msk.
2. $\mathsf{EncSetup}(\mathsf{msk})$*: takes as input the master secret key* msk *and outputs an encryption state* $\mathsf{Enc.st}$
3. $\mathsf{Enc}(\mathsf{msk}, \mathsf{Enc.st}, i, x_i)$*: takes as input the master secret key* msk*, an encryption state* $\mathsf{Enc.st}$*, an index i, and a message $x_i \in \{0,1\}^{\ell_\mathcal{X}}$ and outputs an encryption ct_i of x_i.*
4. $\mathsf{KeyGen}(\mathsf{msk}, f)$*: takes as input the master secret key* msk *and a function $f \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$ and outputs a function key sk_f.*
5. $\mathsf{Dec}(\mathsf{sk}_f, \mathsf{Dec.st}_i, i, \mathsf{ct}_i)$*: where for each function key sk_f, $\mathsf{Dec}(\mathsf{sk}_f, \cdot, \cdot, \cdot)$ is a streaming function that takes as input a state $\mathsf{Dec.st}_i$, an index i, and an encryption ct_i and outputs a new state $\mathsf{Dec.st}_{i+1}$ and an output $y_i \in \{0,1\}^{\ell_\mathcal{Y}}$.*

[15]We also allow $\mathsf{Enc}, \mathsf{EncSetup}, \mathsf{KeyGen}$, and Dec to additionally receive parameters $1^\lambda, 1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{S}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}}$ as input, but omit them from our notation for convenience.

sFE *satisfies **correctness** if for all polynomials p, there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$, all $\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}} \leq p(\lambda)$, all $n \in [2^\lambda]$, all $x = x_1 \dots x_n$ where each $x_i \in \{0,1\}^{\ell_{\mathcal{X}}}$, and all $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$,*

$$\Pr\left[\overline{\mathsf{Dec}}(\mathsf{sk}_f, \mathsf{ct}_x) = f(x) : \begin{array}{c} \mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}}), \\ \mathsf{ct}_x \leftarrow \overline{\mathsf{Enc}}(\mathsf{msk}, x), \\ \mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f) \end{array}\right] \geq 1 - \mu(\lambda)$$

where we define[16]

- *the output of* $\overline{\mathsf{Enc}}(\mathsf{msk}, x)$ *to be* $\mathsf{ct}_x = (\mathsf{ct}_i)_{i \in [n]}$ *produced by sampling* $\mathsf{Enc.st} \leftarrow \mathsf{EncSetup}(\mathsf{msk})$ *and then computing* $ct_i \leftarrow \mathsf{Enc}(\mathsf{msk}, \mathsf{Enc.st}, i, x_i)$ *for* $i \in [n]$.
- *the output of* $\overline{\mathsf{Dec}}(\mathsf{sk}_f, \mathsf{ct}_x)$ *to be* $y = (y_i)_{i \in [n]}$ *where* $(y_i, \mathsf{Dec.st}_{i+1}) = \mathsf{Dec}(\mathsf{sk}_f, \mathsf{Dec.st}_i, i, \mathsf{ct}_i)$

We require the same notion of *streaming efficiency* as with public-key streaming FE.

C.2 Relaxed Definition of Streaming FE

As mentioned in Remark 4.5, we can also consider a relaxed variant of streaming FE in which the encryption function is also a streaming function that takes as input the master public key, a state $\mathsf{Enc.st}_i$, an index i, and an input x_i, and outputs a new state $\mathsf{Enc.st}_{i+1}$, and an encryption ct_i of x_i.

Definition C.2 (Public-Key Streaming FE, Relaxed Definition)**.** *A public-key streaming functional encryption scheme (relaxed definition) for* $\mathsf{P/Poly}$ *is a tuple of PPT algorithms* $\mathsf{sFE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ *defined as follows:*[17]

- $\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}})$*: takes as input the security parameter λ, a function size $\ell_{\mathcal{F}}$, a state size $\ell_{\mathcal{S}}$, an input size $\ell_{\mathcal{X}}$, and an output size $\ell_{\mathcal{Y}}$, and outputs the master public key* mpk *and a master secret key* msk.
- $\mathsf{Enc}(\mathsf{mpk}, \mathsf{Enc.st}_i, i, x_i)$*: where for each master public key* mpk, $\mathsf{Enc}(\mathsf{mpk}, \cdot, \cdot, \cdot)$ *is a (randomized) streaming function that takes as input a state* $\mathsf{Enc.st}_i$*, an index i, and a message $x_i \in \{0,1\}^{\ell_{\mathcal{X}}}$ and outputs a new state* $\mathsf{Enc.st}_{i+1}$ *and an encryption* ct_i *of x_i.*
- $\mathsf{KeyGen}(\mathsf{msk}, f)$*: takes as input the master secret key* msk *and a function $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$ and outputs a function key* sk_f.
- $\mathsf{Dec}(\mathsf{sk}_f, \mathsf{Dec.st}_i, i, \mathsf{ct}_i)$*: where for each function key* sk_f, $\mathsf{Dec}(\mathsf{sk}_f, \cdot, \cdot, \cdot)$ *is a (deterministic) streaming function that takes as input a state* $\mathsf{Dec.st}_i$*, an index i, and an encryption* ct_i *and outputs a new state* $\mathsf{Dec.st}_{i+1}$ *and an output $y_i \in \{0,1\}^{\ell_{\mathcal{Y}}}$.*

sFE *satisfies **correctness** if for all polynomials p, there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$, all $\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}} \leq p(\lambda)$, all $n \in [2^\lambda]$, all $x = x_1 \dots x_n$ where each $x_i \in \{0,1\}^{\ell_{\mathcal{X}}}$, and all $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$,*

$$\Pr\left[\overline{\mathsf{Dec}}(\mathsf{sk}_f, \mathsf{ct}_x) = f(x) : \begin{array}{c} (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}}), \\ \mathsf{ct}_x \leftarrow \overline{\mathsf{Enc}}(\mathsf{mpk}, x), \\ \mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f) \end{array}\right] \geq 1 - \mu(\lambda)$$

where we define[18]

[16] As with all streaming functions, we assume that $\mathsf{Dec.st}_1 = \bot$ by default.

[17] We also allow $\mathsf{Enc}, \mathsf{KeyGen}$, and Dec to additionally receive parameters $1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}}$ as input, but omit them from our notation for convenience.

[18] As with all streaming functions, we assume that $\mathsf{Enc.st}_1 = \mathsf{Dec.st}_1 = \bot$ by default.

- *the output of* $\overline{\mathsf{Enc}}(\mathsf{mpk}, x)$ *to be* $\mathsf{ct}_x = (\mathsf{ct}_i)_{i\in[n]}$ *where* $(\mathsf{ct}_i, \mathsf{Enc.st}_{i+1}) \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{Enc.st}_i, i, x_i)$
- *the output of* $\overline{\mathsf{Dec}}(\mathsf{sk}_f, \mathsf{ct}_x)$ *to be* $y = (y_i)_{i\in[n]}$ *where* $(y_i, \mathsf{Dec.st}_{i+1}) = \mathsf{Dec}(\mathsf{sk}_f, \mathsf{Dec.st}_i, i, \mathsf{ct}_i)$

We require the same notion of *streaming efficiency* as in the original definition of public-key streaming FE. The definitions of security are the same as before except that we define $\overline{\mathsf{Enc}}(\mathsf{mpk}, x)$ according to Definition C.2. We can also similarly define this in the secret-key setting, where Setup only outputs a master secret key and Enc only requires the master secret key instead of the (non-existent) master public key.

Remark C.3. Observe that a public-key streaming FE is a special case of the relaxed definition of a public-key streaming FE. If $\mathsf{sFE} = (\mathsf{Setup}, \mathsf{EncSetup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is a public-key streaming FE scheme, then we can create a public-key streaming FE scheme $\mathsf{sFE}' = (\mathsf{Setup}, \mathsf{Enc}', \mathsf{KeyGen}, \mathsf{Dec})$ according to our relaxed definition where we define

```
$\mathsf{Enc}'(\mathsf{mpk}, \mathsf{Enc}'.\mathsf{st}_i, i, x_i)$:
    1. If $i = 1$,
        (a) $\mathsf{Enc.st} \leftarrow \mathsf{EncSetup}(\mathsf{mpk})$
        (b) $\mathsf{ct}_1 \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{Enc.st}, 1, x_1)$
        (c) Output $(\mathsf{ct}_1, \mathsf{Enc}'.\mathsf{st}_2 = \mathsf{Enc.st})$
    2. Else
        (a) Parse $\mathsf{Enc}'.\mathsf{st}_i = \mathsf{Enc.st}$
        (b) $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{Enc.st}, i, x_i)$
        (c) Output $(\mathsf{ct}_i, \mathsf{Enc}'.\mathsf{st}_{i+1} = \mathsf{Enc.st})$
```

D Security Proof from Section 6

In this section, we prove that sFE from Section 6 is semi-adaptive-function-selective-IND-secure (see Definition 4.6). In this proof, we will use an alternate, but equivalent definition of semi-adaptive-function-selective-IND-security.

Definition D.1 (Semi-Adaptive-Function-Selective-IND-Security, Equivalent Definition)**.** *A public-key streaming FE scheme* sFE *for* P/Poly *is semi-adaptive-function-selective-IND-secure if there exists a negligible function* μ *such that for all* $\lambda \in \mathbb{N}$ *and all PPT adversaries* $\mathcal{A}$*,*

$$\left|\Pr[\mathsf{ExptGuess}_{\mathcal{A}}^{\mathsf{Semi\text{-}Ad\text{-}Func\text{-}Sel\text{-}IND}}(1^\lambda) = 1]\right| \le \frac{1}{2} + \mu(\lambda)$$

where for each $\lambda \in \mathbb{N}$*, we define*

$\mathsf{ExptGuess}_{\mathcal{A}}^{\mathsf{Semi\text{-}Ad\text{-}Func\text{-}Sel\text{-}IND}}(1^\lambda)$

1. ***Parameters:*** *$\mathcal{A}$ takes as input 1^λ, and outputs a function size $1^{\ell_\mathcal{F}}$, a state size $1^{\ell_\mathcal{S}}$, an input size $1^{\ell_\mathcal{X}}$, and an output size $1^{\ell_\mathcal{Y}}$.*

2. ***Public Key:*** *Compute* $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{sFE.Setup}(1^\lambda, 1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{S}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}})$ *and send* mpk *to* $\mathcal{A}$*.*

3. ***Function Queries:*** *The following can be repeated any polynomial number of times:*

   (a) $\mathcal{A}$ *outputs a function query* $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$

   (b) $\mathsf{sk}_f \leftarrow \mathsf{sFE.KeyGen}(\mathsf{msk}, f)$

   (c) *Send* sk_f *to* $\mathcal{A}$

4. ***Challenge Message:*** $\mathcal{A}$ *outputs a challenge message pair* $(x^{(0)}, x^{(1)})$ *where* $x^{(0)} = x_1^{(0)} \dots x_n^{(0)}$ *and* $x^{(1)} = x_1^{(1)} \dots x_n^{(1)}$ *for some length* $n \in \mathbb{N}$ *chosen by the adversary and where each* $x_i^{(0)}, x_i^{(1)} \in \{0,1\}^{\ell_{\mathcal{X}}}$.

5. ***Challenge Bit:*** *Sample* $b \leftarrow \{0,1\}$.

6. ***Challenge Ciphertext:*** *Compute* $\mathsf{ct} \leftarrow \mathsf{sFE}.\overline{\mathsf{Enc}}(\mathsf{mpk}, x^{(b)})$ *and send* ct *to* $\mathcal{A}$.

7. ***Experiment Outcome:*** $\mathcal{A}$ *outputs a bit* b'. *The output of the experiment is set to* 1 *if* $b = b'$ *and* $f(x^{(0)}) = f(x^{(1)})$ *for all functions* f *queried by the adversary.*

This is equivalent to the regular definition as for any adversary $\mathcal{A}$,

$$\left|\Pr[\mathsf{ExptGuess}_{\mathcal{A}}^{\mathsf{Semi\text{-}Ad\text{-}Func\text{-}Sel\text{-}IND}}(1^\lambda) = 1]\right| \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

if and only if

$$\left|\Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{Semi\text{-}Ad\text{-}Func\text{-}Sel\text{-}IND}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Expt}_{\mathcal{A}}^{\mathsf{Semi\text{-}Ad\text{-}Func\text{-}Sel\text{-}IND}}(1^\lambda, 1) = 1]\right| \leq \mathsf{negl}(\lambda)$$

Notation. Recall that for notational convenience, we may omit the security, input size, output size, message size, function size, or state size parameters from our algorithms. For information on these parameters, please see the parameter section in Section 6.

D.0.1 Proof Overview

To build intuition, we provide a brief overview of each hybrid below.

- **Hybrid**$_1^{\mathcal{A}}$: This is the real world experiment. The adversary first receives the security parameter and chooses the function size, state size, input size, and output size. Then, the adversary receives the master public key MPK. After that, the adversary can adaptively receive function keys sk_{f_j} for streaming functions f_j of its choice. Next, the adversary submits a challenge message pair $(x^{(0)}, x^{(1)})$ and receives a ciphertext of $x^{(b)}$ for a random bit $b \in \{0,1\}$. The adversary guesses b and wins if it guesses b correctly and if $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried.

- **Hybrid**$_2^{\mathcal{A}}$: We hardcode in values for the $\alpha = 1$ branch of G_{f_j,s_j,c_j} for each function key. For each function query f_j, we hardcode into c_j the values $(\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j)$ that are output by G_{f_j,s_j,c_j} on the $\alpha = 0$ branch if we run it on the input $(\mathsf{FPFE.msk}, \mathsf{PRF}.K, 0, 0^{\ell_{\mathsf{Sym}.k_\lambda}})$ generated by the challenge message. Note that this input is independent of the choice of challenge messages $(x^{(0)}, x^{(1)})$. (By hardcode, we mean that we generate $c_i \leftarrow \mathsf{Sym.Enc}(\mathsf{Sym}.k, (\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j)))$. The objective is to use the security of FE in the next hybrid to switch to the $\alpha = 1$ branch of each G_{f_j,c_j,s_j}, which does not require knowledge of $\mathsf{PRF}.K$ or $\mathsf{FPFE.msk}$ in the input. As $\mathsf{PRF}.K$ is used to generate all of the $\mathsf{One\text{-}sFE}$ master secret keys, being able to remove this value will allow us to hide these $\mathsf{One\text{-}sFE}$

master secret keys in later hybrids. The indistinguishability of $\mathbf{Hybrid}_1^{\mathcal{A}}$ and $\mathbf{Hybrid}_2^{\mathcal{A}}$ holds by the pseudorandom ciphertext property of Sym.

- $\mathbf{Hybrid}_3^{\mathcal{A}}$: In the challenge ciphertext, instead of encrypting

$$\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (\mathsf{FPFE.msk}, \mathsf{PRF}.K, 0, 0^{\ell_{\mathsf{Sym}.k_\lambda}}))$$

we encrypt

$$\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (0^{\ell_{\mathsf{FPFE.msk}_\lambda}}, 0^{\ell_{\mathsf{PRF}.K_\lambda}}, 1, \mathsf{Sym}.k))$$

Observe that the only functions keys generated using the corresponding $\mathsf{FE.msk}$ are for functions G_{f_j,s_j,c_j}. However, because we have hardcoded the correct output values into each c_j in our previous hybrid, then for all j,

$$G_{f_j,s_j,c_j}(\mathsf{FPFE.msk}, \mathsf{PRF}.K, 0, 0^{\ell_{\mathsf{Sym}.k_\lambda}}) = G_{f_j,s_j,c_j}(0^{\ell_{\mathsf{FPFE.msk}_\lambda}}, 0^{\ell_{\mathsf{PRF}.K_\lambda}}, 1, \mathsf{Sym}.k)$$

Thus, the indistinguishability of $\mathbf{Hybrid}_2^{\mathcal{A}}$ and $\mathbf{Hybrid}_3^{\mathcal{A}}$ holds by the selective-IND-security of FE. Selective security is sufficient as the messages $(\mathsf{FPFE.msk}, \mathsf{PRF}.K, 0, 0^{\ell_{\mathsf{Sym}.k_\lambda}})$ and $(0^{\ell_{\mathsf{FPFE.msk}_\lambda}}, 0^{\ell_{\mathsf{PRF}.K_\lambda}}, 1, \mathsf{Sym}.k)$ can be computed at the beginning of the experiment, even before learning $\mathsf{FE.mpk}$.

- $\mathbf{Hybrid}_4^{\mathcal{A}}$: For each j, to determine the values we need to hardcode into c_j, we use randomness $r_{\mathsf{Setup},j}, r_{\mathsf{KeyGen},j}, r_{\mathsf{EncSetup},j}, r_{\mathsf{PRF2},j}, r_{\mathsf{Enc},j}$ to generate $\mathsf{One\text{-}sFE.msk}_j$, $\mathsf{One\text{-}sFE.Enc.st}_j$, $\mathsf{One\text{-}sFE.sk}_{f_j}$, $\mathsf{PRF2}.k_j$, and $\mathsf{FPFE.ct}_j$. Instead of generating these random values using $\mathsf{PRF}.K$, we now generate these values using true randomness. Because of the change made in our previous hybrid, the key $\mathsf{PRF}.K$ is not used anywhere else in our experiment, so the indistinguishability of $\mathbf{Hybrid}_3^{\mathcal{A}}$ and $\mathbf{Hybrid}_4^{\mathcal{A}}$ holds by the security of PRF.

- $\mathbf{Hybrid}_5^{\mathcal{A}}$: In the ciphertext, we replace the FPFE function keys for $H_{i,x_i^{(b)},t_i}$ with function keys for new functions $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_i}$ (defined in Figure 11) that have additional branches of computation.
  - When $\beta = 0$, $H^*_{i,x_i^{(b)},x_i^{(0)},v_i}$ will act the same as $H_{i,x_i^{(b)},t_i}$ and will generate a $\mathsf{One\text{-}sFE}$ ciphertext for $x_i^{(b)}$.
  - When $\beta = 1$, $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_i}$ will instead generate a $\mathsf{One\text{-}sFE}$ ciphertext for $x_i^{(0)}$.
  - When $\beta = 2$, $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_i}$ will simply output v_i (which is set to 0 in this hybrid).

  As $H_{i,x_i^{(b)},t_i}$ and $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_i}$ act the same when $\beta = 0$, and we only encrypt FPFE messages where $\beta = 0$, then the indistinguishability of $\mathbf{Hybrid}_4^{\mathcal{A}}$ and $\mathbf{Hybrid}_5^{\mathcal{A}}$ holds by the function privacy of FPFE.

- We will now go through a series of hybrids for $k = 1$ to q where $q = q(\lambda)$ is the runtime of $\mathcal{A}$ and an implicit bound on the number of function queries made by $\mathcal{A}$. At a high level, the goal is to one by one switch to the $\beta = 1$ branch in every FPFE ciphertext. This will allow us to use the function privacy of FPFE to remove the dependence on b present in the $\beta = 0$ branch of each $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_i}$.

– $\textbf{Hybrid}^{\mathcal{A}}_{6,k,1}$: We prepare to switch to the $\beta = 2$ branch in the k^{th} FPFE ciphertext. For each i, we replace the value v_i in the FPFE function key of $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_i}$ (or for $k > 1$, the value $v_{i,k-1}$ in $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k-1}}$) with a new value $v_{i,k}$ which corresponds to the output of $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_i}$ on the message $(\mathsf{One\text{-}sFE.msk}_k, \mathsf{One\text{-}sFE.Enc.st}_k, \mathsf{PRF2}.k_k, 0)$ encrypted in the k^{th} FPFE ciphertext. This value $v_{i,k}$ is an encryption of $x_i^{(b)}$ under $\mathsf{One\text{-}sFE.msk}_k$ using randomness generated by $\mathsf{PRF2}.k_k$. Since the value of v_i (or $v_{i,k-1}$) only affects the $\beta = 2$ branch of $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_i}$ (or $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k-1}}$), and we only encrypt FPFE ciphertexts where $\beta = 0$ or $\beta = 1$, then we can perform this change due to the function privacy of FPFE.

– $\textbf{Hybrid}^{\mathcal{A}}_{6,k,2}$: We now switch to the $\beta = 2$ branch of the k^{th} FPFE ciphertext. When we hardcode values into c_k in our function key, instead of encrypting

$$\mathsf{FPFE.ct}_k \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_k, \mathsf{One\text{-}sFE.Enc.st}_k, \mathsf{PRF2}.k_k, 0))$$

we encrypt

$$\mathsf{FPFE.ct}_k \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (0^{\ell_{\mathsf{One\text{-}sFE.msk}_\lambda}}, 0^{\ell_{\mathsf{One\text{-}sFE.Enc.st}_\lambda}}, 0^{\ell_{\mathsf{PRF2}.k_\lambda}}, 2))$$

Observe that the only FPFE function keys generated using $\mathsf{FPFE.msk}$ are for functions $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}$. However, because we hardcoded the correct output values into each $v_{i,k}$, then

$$\begin{aligned} &H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}(\mathsf{One\text{-}sFE.msk}_k, \mathsf{One\text{-}sFE.Enc.st}_k, \mathsf{PRF2}.k_k, 0) \\ &= H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}(0^{\ell_{\mathsf{One\text{-}sFE.msk}_\lambda}} 0^{\ell_{\mathsf{One\text{-}sFE.Enc.st}_\lambda}}, 0^{\ell_{\mathsf{PRF2}.k_\lambda}}, 2) \end{aligned}$$

Thus, the indistinguishability of $\textbf{Hybrid}^{\mathcal{A}}_{6,k,1}$ and $\textbf{Hybrid}^{\mathcal{A}}_{6,k,2}$ holds by the message privacy of FPFE.

– $\textbf{Hybrid}^{\mathcal{A}}_{6,k,3}$: We would now like to change $v_{i,k}$ from a $\mathsf{One\text{-}sFE}$ encryption of $x_i^{(b)}$ to a $\mathsf{One\text{-}sFE}$ encryption of $x_i^{(0)}$. However, in order to perform that step, we first need to use true randomness for the encryption. Thus, in this hybrid, instead of generating $r_{i,k}$ (which is the randomness used to generate $v_{i,k}$: the i^{th} ciphertext of $x_i^{(b)}$ under $\mathsf{One\text{-}sFE.msk}_k$ and $\mathsf{One\text{-}sFE.Enc.st}_k$) using $\mathsf{PRF2}.k_k$, we generate $r_{i,k}$ using true randomness. Observe that $\mathsf{PRF2}.k_k$ was removed from our experiment in the previous hybrid when we switched to the $\beta = 2$ branch in $\mathsf{FPFE.ct}_k$. Thus, the indistinguishability of $\textbf{Hybrid}^{\mathcal{A}}_{6,k,2}$ and $\textbf{Hybrid}^{\mathcal{A}}_{6,k,3}$ holds by the security of PRF2.

– $\textbf{Hybrid}^{\mathcal{A}}_{6,k,4}$: We now invoke the security of $\mathsf{One\text{-}sFE}$ to change the value of $v_{i,k}$. For each i, instead of computing

$$v_{i,k} \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}_k, \mathsf{One\text{-}sFE.Enc.st}_k, i, x_i^{(b)})$$

we compute

$$v_{i,k} \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}_k, \mathsf{One\text{-}sFE.Enc.st}_k, i, x_i^{(0)})$$

Observe that this is equivalent to switching from an encryption of $x^{(b)}$ under $\mathsf{One\text{-}sFE.msk}_k$ to an encryption of $x^{(0)}$ under $\mathsf{One\text{-}sFE.msk}_k$. (If for $d \in \{0,1\}$, $\mathsf{CT}^{(d)} = \{\mathsf{CT}_i^{(d)}\}_{i\in[n]}$

is an encryption of $x^{(d)}$ under $\mathsf{One\text{-}sFE.msk}_k$, then $v_{i,k} = \mathsf{CT}_i^{(b)}$ in the former case and $v_{i,k} = \mathsf{CT}_i^{(0)}$ in the latter.) To allow this change under the single-key, single-ciphertext, function-selective-IND-security of One-sFE, we need to ensure the following:

1. We only use $\mathsf{One\text{-}sFE.msk}_k$ and $\mathsf{One\text{-}sFE.Enc.st}_k$ for one ciphertext and one function key. For our challenge message, every function query generates a different One-sFE master secret key. Thus, we only use these values for one ciphertext (namely the challenge ciphertext) and one key (corresponding to the k^{th} function query f_k).
2. The One-sFE challenge function f_k has the same output value on the challenge messages $x^{(b)}$ and $x^{(0)}$. This holds since the sFE security game requires $f_j(x^{(0)}) = f_j(x^{(1)})$ for all functions f_j queried, so indeed $f_k(x^{(b)}) = f_k(x^{(0)})$.
3. We ask for the challenge function key before the challenge ciphertext. This can be easily observed in the hybrid.
4. We do not leak additional information about $\mathsf{One\text{-}sFE.msk}_k$, $\mathsf{One\text{-}sFE.Enc.st}_k$, or the randomness used to generate the ciphertext or function key. Except for their appearances in the k^{th} One-sFE ciphertext and function key, the only place that $\mathsf{One\text{-}sFE.msk}_k$ and $\mathsf{One\text{-}sFE.Enc.st}_k$ appeared was in $\mathsf{FPFE.ct}_k$. However, we removed these values from $\mathsf{FPFE.ct}_k$ in a previous hybrid when we switched to the $\beta = 2$ branch. Observe also that the randomness used is independent and uniform as we have already removed $\mathsf{PRF}.K$ and $\mathsf{PRF2}.k_k$ from the experiment.

Thus, the indistinguishability of $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,3}$ and $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,4}$ holds by the security of One-sFE.

– $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,5}$: We undo the change made in $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,3}$. Instead of computing $v_{i,k}$ using true randomness, we compute $v_{i,k}$ using randomness $r_{i,k}$ generated by the k^{th} PRF2 key $\mathsf{PRF2}.k_k$. The indistinguishability of $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,4}$ and $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,5}$ holds by the security of PRF2.

– $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,6}$: We now switch to the $\beta = 1$ branch in the k^{th} ciphertext. When we hardcode values into c_k in our function key, instead of encrypting

$$\mathsf{FPFE.ct}_k \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (0^{\ell_{\mathsf{One\text{-}sFE.msk}_\lambda}}, 0^{\ell_{\mathsf{One\text{-}sFE.Enc.st}_\lambda}}, 0^{\ell_{\mathsf{PRF2}.k_\lambda}}, 2))$$

we encrypt

$$\mathsf{FPFE.ct}_k \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_k, \mathsf{One\text{-}sFE.Enc.st}_k, \mathsf{PRF2}.k_k, 1))$$

Observe that the only FPFE function keys we generated using FPFE.msk are for functions $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}$. However, we observe that the value of $v_{i,k}$ is now in fact equal to $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}(\mathsf{One\text{-}sFE.msk}_k, \mathsf{One\text{-}sFE.Enc.st}_k, \mathsf{PRF2}.k_k, 1)$ as it is an encryption of $x_i^{(0)}$ under $\mathsf{One\text{-}sFE.msk}_k$ using randomness generated by $\mathsf{PRF2}.k_k$. Therefore,

$$\begin{aligned} &H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}(\mathsf{One\text{-}sFE.msk}_k, \mathsf{One\text{-}sFE.Enc.st}_k, \mathsf{PRF2}.k_k, 1) \\ &= H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}(0^{\ell_{\mathsf{One\text{-}sFE.msk}_\lambda}}, 0^{\ell_{\mathsf{One\text{-}sFE.Enc.st}_\lambda}} 0^{\ell_{\mathsf{PRF2}.k_\lambda}}, 2) \end{aligned}$$

and the indistinguishability of $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,5}$ and $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,6}$ holds by the message privacy of FPFE.

- **Hybrid**$_7^{\mathcal{A}}$: In the ciphertext, we replace the FPFE function keys for $H^*_{i,x_i^{(b)},x_i^{(0)},t_i v_{i,q}}$ (where q is the runtime of $\mathcal{A}$) with FPFE function keys for functions $H^*_{i,x_i^{(0)},x_i^{(0)},t_i,v_i}$ where v_i is set to 0. Observe that q is an implicit bound on the number of function queries made by $\mathcal{A}$ and thus on the number of FPFE ciphertexts that we generate. Therefore, by the time we reach **Hybrid**$_{6,q,6}^{\mathcal{A}}$, we will have switched all FPFE ciphertexts to the $\beta = 1$ branch. But since $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,q}}$ and $H^*_{i,x_i^{(0)},x_i^{(0)},t_i,v_i}$ act the same when $\beta = 1$, then the indistinguishability of **Hybrid**$_{6,q,6}^{\mathcal{A}}$ and **Hybrid**$_7$ holds by the function privacy of FPFE.

Our final hybrid **Hybrid**$_7^{\mathcal{A}}$ is independent of the bit b. Thus, any adversary's advantage in guessing b in **Hybrid**$_7^{\mathcal{A}}$ is zero. But our proof shows that for any PPT adversary $\mathcal{A}$, $\mathcal{A}$'s advantage in guessing b in **Hybrid**$_1^{\mathcal{A}}$ is negligibly close to $\mathcal{A}$'s advantage in guessing b in **Hybrid**$_7^{\mathcal{A}}$. Thus, for any PPT adversary $\mathcal{A}$, the advantage in guessing b in the real world must be negligible, so security holds.

D.0.2 Formal Proof

We now formally prove security via a hybrid argument.

Hybrid$_1^{\mathcal{A}}(1^\lambda)$: This is the real world experiment. Though we have reordered some steps for the sake of the proof, this does not affect the outcome of the experiment.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs a function size $1^{\ell_\mathcal{F}}$, a state size $1^{\ell_\mathcal{S}}$, an input size $1^{\ell_\mathcal{X}}$, and an output size $1^{\ell_\mathcal{Y}}$.
2. **Setup:**
   (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$
   (b) $\mathsf{PRF}.K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$
   (c) $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$
   (d) $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (\mathsf{FPFE.msk}, \mathsf{PRF}.K, 0, 0^{\ell_{\mathsf{Sym}.k_\lambda}}))$
3. **Public Key:** Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.
4. **Function Queries:** For the j^{th} function query $f_j \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$ made by the adversary:
   (a) $s_j \leftarrow \{0,1\}^\lambda$
   (b) $c_j \leftarrow \{0,1\}^{\ell_{\mathsf{Sym.ct}_\lambda}}$
   (c) Let $G_j = G_{f_j, s_j, c_j}$ as defined in Figure 9.
   (d) $\mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_j)$
   (e) Send $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to the adversary.
5. **Challenge Message:** $\mathcal{A}$ outputs a challenge message pair $(x^{(0)}, x^{(1)})$ where $x^{(0)} = x_1^{(0)} \dots x_n^{(0)}$ and $x^{(1)} = x_1^{(1)} \dots x_n^{(1)}$ for some length $n \in \mathbb{N}$ chosen by the adversary and where each $x_i^{(0)}, x_i^{(1)} \in \{0,1\}^{\ell_\mathcal{X}}$.
6. **Challenge Bit:** $b \leftarrow \{0,1\}$
7. **Challenge Ciphertext:**
   (a) For $i \in [n]$,
      i. $t_i \leftarrow \{0,1\}^\lambda$
      ii. Let $H_i = H_{i, x_i^{(b)}, t_i}$ as defined in Figure 8.
      iii. $\mathsf{FPFE.sk}_{H_i} = \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H_i)$
      iv. If $i = 1$, let $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})$. Else, let $\mathsf{CT}_i = \mathsf{FPFE.sk}_{H_i}$
   (b) Send $\mathsf{CT} = \{\mathsf{CT}_i\}_{i \in [n]}$ to the adversary.
8. **Experiment Outcome:** The adversary outputs a bit b'. Output 1 if $b = b'$ and $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried, and output 0 otherwise.

Hybrid$_2^{\mathcal{A}}(1^\lambda)$: For each j, we hardcode into c_j the values

$$(\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j) = G_{f_j,s_j,c_j}(\mathsf{FPFE.msk}, \mathsf{PRF}.K, 0, 0^{\ell_{\mathsf{Sym}.k_\lambda}})$$

which would be generated in the real world experiment. This will allow us to later switch to the $\alpha = 1$ branch in G_{f_j,s_j,c_j} using the security of FE. Observe that the values being hardcoded into c_j can be computed before knowing $x^{(0)}$ or $x^{(1)}$.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs a function size $1^{\ell_\mathcal{F}}$, a state size $1^{\ell_\mathcal{S}}$, an input size $1^{\ell_\mathcal{X}}$, and an output size $1^{\ell_\mathcal{Y}}$.

2. **Setup:**

   (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$

   (b) $\mathsf{PRF}.K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$

   (c) $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$

   (d) $\mathsf{Sym}.k \leftarrow \mathsf{Sym.Setup}(1^\lambda)$

   (e) $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (\mathsf{FPFE.msk}, \mathsf{PRF}.K, 0, 0^{\ell_{\mathsf{Sym}.k_\lambda}}))$

3. **Public Key:** Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.

4. **Function Queries:** For the j^{th} function query $f_j \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$ made by the adversary:

   (a) $s_j \leftarrow \{0,1\}^\lambda$

   (b) **Compute c_j:**

      i. $(r_{\mathsf{Setup},j}, r_{\mathsf{KeyGen},j}, r_{\mathsf{EncSetup},j}, r_{\mathsf{PRF2},j}, r_{\mathsf{Enc},j}) \leftarrow \mathsf{PRF.Eval}(\mathsf{PRF}.K, s_j)$

      ii. $\mathsf{One\text{-}sFE.msk}_j \leftarrow \mathsf{One\text{-}sFE.Setup}(1^\lambda; r_{\mathsf{Setup},j})$

      iii. $\mathsf{One\text{-}sFE.Enc.st}_j \leftarrow \mathsf{One\text{-}sFE.EncSetup}(\mathsf{One\text{-}sFE.msk}_j; r_{\mathsf{EncSetup}_j})$

      iv. $\mathsf{One\text{-}sFE.sk}_{f_j} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_j, f_j; r_{\mathsf{KeyGen},j})$

      v. $\mathsf{PRF2}.k_j \leftarrow \mathsf{PRF2.Setup}(1^\lambda; r_{\mathsf{PRF2},j})$

      vi. $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0); r_{\mathsf{Enc},j})$

      vii. $c_j \leftarrow \mathsf{Sym.Enc}(\mathsf{Sym}.k, (\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j))$

   (c) Let $G_j = G_{f_j,s_j,c_j}$ as defined in Figure 9.

   (d) $\mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_j)$

   (e) Send $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to the adversary.

5. **Challenge Message:** $\mathcal{A}$ outputs a challenge message pair $(x^{(0)}, x^{(1)})$ where $x^{(0)} = x_1^{(0)} \dots x_n^{(0)}$ and $x^{(1)} = x_1^{(1)} \dots x_n^{(1)}$ for some length $n \in \mathbb{N}$ chosen by the adversary and where each $x_i^{(0)}, x_i^{(1)} \in \{0,1\}^{\ell_\mathcal{X}}$.

6. **Challenge Bit:** $b \leftarrow \{0,1\}$

7. **Challenge Ciphertext:**

   (a) For $i \in [n]$,

      i. $t_i \leftarrow \{0,1\}^\lambda$

      ii. Let $H_i = H_{i,x_i^{(b)},t_i}$ as defined in Figure 8.

iii. $\mathsf{FPFE.sk}_{H_i} = \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H_i)$

iv. If $i = 1$, let $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})$. Else, let $\mathsf{CT}_i = \mathsf{FPFE.sk}_{H_i}$

(b) Send $\mathsf{CT} = \{\mathsf{CT}_i\}_{i \in [n]}$ to the adversary.

8. **Experiment Outcome:** The adversary outputs a bit b'. Output 1 if $b = b'$ and $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried, and output 0 otherwise.

Lemma D.2. *If* Sym *has pseudorandom ciphertexts, then for all PPT adversaries $\mathcal{A}$,*

$$\left|\Pr[\mathbf{Hybrid}_1^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_2^{\mathcal{A}}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

Proof. Suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$ such that

$$\left|\Pr[\mathbf{Hybrid}_1^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_2^{\mathcal{A}}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{9}$$

We build a PPT adversary $\mathcal{B}$ that breaks the pseudorandom ciphertext property of Sym. $\mathcal{B}$ first runs $\mathcal{A}$ on input 1^λ and receives parameters $1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{S}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}}$. $\mathcal{B}$ then sends message length $1^{\ell_{\mathsf{Sym}.m_\lambda}}$ to its Sym challenger where where $\ell_{\mathsf{Sym}.m_\lambda}$ is computed as described in the parameter section. $\mathcal{B}$ then computes $(\mathsf{FE.mpk}, \mathsf{FE.msk}, \mathsf{PRF}.K, \mathsf{FPFE.msk}, \mathsf{FE.ct})$ as in $\mathbf{Hybrid}_1^{\mathcal{A}}$ and sends $\mathsf{MPK} = \mathsf{FE.mpk}$ to $\mathcal{A}$. For each function query f_j that $\mathcal{A}$ sends to $\mathcal{B}$, $\mathcal{B}$ does the following: $\mathcal{B}$ computes $s_j \leftarrow \{0,1\}^\lambda$ and $(\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j)$ as in $\mathbf{Hybrid}_2^{\mathcal{A}}$. $\mathcal{B}$ sends $(\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j)$ as its challenge message to its Sym challenger and receives c_j which is either a uniform random value or an encryption of $(\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j)$ under Sym. $\mathcal{B}$ then computes $\mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_{f_j,s_j,c_j})$ and sends $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to $\mathcal{A}$. After $\mathcal{A}$ is done making function queries, $\mathcal{A}$ outputs challenge messages $(x^{(0)}, x^{(1)})$. $\mathcal{B}$ samples $b \leftarrow \{0,1\}$, computes CT as in $\mathbf{Hybrid}_1^{\mathcal{A}}$, sends CT to $\mathcal{A}$, and receives b' from $\mathcal{A}$. $\mathcal{B}$ outputs 1 if $b = b'$ and $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried, and outputs 0 otherwise. Observe that if every c_j is an independent uniform random value, then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}_1^{\mathcal{A}}$, and if each c_j is an encryption of $\mathcal{B}$'s j^{th} challenge message $(\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j)$ under Sym, then $\mathcal{B}$ emulates $\mathbf{Hybrid}_2^{\mathcal{A}}$. Additionally, $\mathcal{B}$ does not need to know $\mathsf{Sym}.k$ to carry out this experiment. Thus, by Equation 9, this means that $\mathcal{B}$ breaks the pseudorandom ciphertext property of Sym as $\mathcal{B}$ can distinguish between receiving random values and valid ciphertexts with non-negligible probability. □

Hybrid$_3^{\mathcal{A}}$: We change the message encrypted in $\mathsf{FE.ct}$ so that we use the $\alpha = 1$ branch of every G_{f_j,s_j,c_j}. This allows us to remove $\mathsf{FPFE.msk}$ and $\mathsf{PRF}.K$ from $\mathsf{FE.ct}$.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs a function size $1^{\ell_\mathcal{F}}$, a state size $1^{\ell_\mathcal{S}}$, an input size $1^{\ell_\mathcal{X}}$, and an output size $1^{\ell_\mathcal{Y}}$.
2. **Setup:**
   (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$
   (b) $\mathsf{PRF}.K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$
   (c) $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$
   (d) $\mathsf{Sym}.k \leftarrow \mathsf{Sym.Setup}(1^\lambda)$
   (e) $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (0^{\ell_{\mathsf{FPFE.msk}_\lambda}}, 0^{\ell_{\mathsf{PRF}.k_\lambda}}, 1, \mathsf{Sym}.k))$
3. **Public Key:** Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.
4. **Function Queries:** For the j^{th} function query $f_j \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$ made by the adversary:
   (a) $s_j \leftarrow \{0,1\}^\lambda$
   (b) **Compute c_j:**
      i. $(r_{\mathsf{Setup},j}, r_{\mathsf{KeyGen},j}, r_{\mathsf{EncSetup}_j}, r_{\mathsf{PRF2},j}, r_{\mathsf{Enc},j}) \leftarrow \mathsf{PRF.Eval}(\mathsf{PRF}.K, s_j)$
      ii. $\mathsf{One\text{-}sFE.msk}_j \leftarrow \mathsf{One\text{-}sFE.Setup}(1^\lambda; r_{\mathsf{Setup},j})$
      iii. $\mathsf{One\text{-}sFE.Enc.st}_j \leftarrow \mathsf{One\text{-}sFE.EncSetup}(\mathsf{One\text{-}sFE.msk}_j; r_{\mathsf{EncSetup}_j})$
      iv. $\mathsf{One\text{-}sFE.sk}_{f_j} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_j, f_j; r_{\mathsf{KeyGen},j})$
      v. $\mathsf{PRF2}.k_j \leftarrow \mathsf{PRF2.Setup}(1^\lambda; r_{\mathsf{PRF2},j})$
      vi. $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0); r_{\mathsf{Enc},j})$
      vii. $c_j \leftarrow \mathsf{Sym.Enc}(\mathsf{Sym}.k, (\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j))$
   (c) Let $G_j = G_{f_j,s_j,c_j}$ as defined in Figure 9.
   (d) $\mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_j)$
   (e) Send $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to the adversary.
5. **Challenge Message:** $\mathcal{A}$ outputs a challenge message pair $(x^{(0)}, x^{(1)})$ where $x^{(0)} = x_1^{(0)} \dots x_n^{(0)}$ and $x^{(1)} = x_1^{(1)} \dots x_n^{(1)}$ for some length $n \in \mathbb{N}$ chosen by the adversary and where each $x_i^{(0)}, x_i^{(1)} \in \{0,1\}^{\ell_\mathcal{X}}$.
6. **Challenge Bit:** $b \leftarrow \{0,1\}$
7. **Challenge Ciphertext:**
   (a) For $i \in [n]$,
      i. $t_i \leftarrow \{0,1\}^\lambda$
      ii. Let $H_i = H_{i,x_i^{(b)},t_i}$ as defined in Figure 8.
      iii. $\mathsf{FPFE.sk}_{H_i} = \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H_i)$
      iv. If $i = 1$, let $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})$. Else, let $\mathsf{CT}_i = \mathsf{FPFE.sk}_{H_i}$
   (b) Send $\mathsf{CT} = \{\mathsf{CT}_i\}_{i\in[n]}$ to the adversary.

8. **Experiment Outcome:** The adversary outputs a bit b'. Output 1 if $b = b'$ and $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried, and output 0 otherwise.

Lemma D.3. *If* FE *is selectively IND-secure, then for all PPT adversaries,*

$$\left|\Pr[\mathbf{Hybrid}_2^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_3^{\mathcal{A}}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

Proof. Suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$ such that

$$\left|\Pr[\mathbf{Hybrid}_2^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_3^{\mathcal{A}}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{10}$$

We build a PPT adversary $\mathcal{B}$ that breaks the selective-IND-security of FE. $\mathcal{B}$ first runs $\mathcal{A}$ on input 1^λ and receives parameters $1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{S}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}}$. $\mathcal{B}$ then sends function size $1^{\ell_{G_\lambda}}$, input size $1^{\ell_{\mathsf{FE}.m_\lambda}}$, and output size $1^{\ell_{\mathsf{FE}.out_\lambda}}$ to its FE challenger where $\ell_{G_\lambda}, \ell_{\mathsf{FE}.m_\lambda}, \ell_{\mathsf{FE}.out_\lambda}$ are computed as described in the parameter section. $\mathcal{B}$ computes $(\mathsf{PRF}.K, \mathsf{FPFE.msk}, \mathsf{Sym}.k)$ as in $\mathbf{Hybrid}_2^{\mathcal{A}}$. $\mathcal{B}$ sends challenge message pair $((\mathsf{FPFE.msk}, \mathsf{PRF}.K, 0, 0^{\ell_{\mathsf{Sym}.k_\lambda}}), (0^{\ell_{\mathsf{FPFE.msk}_\lambda}}, 0^{\ell_{\mathsf{PRF}.K_\lambda}}, 1, \mathsf{Sym}.k))$ to its FE challenger and receives $(\mathsf{FE.mpk}, \mathsf{FE.ct})$ where $\mathsf{FE.ct}$ is either an encryption of $(\mathsf{FPFE.msk}, \mathsf{PRF}.K, 0, 0^{\ell_{\mathsf{Sym}.k_\lambda}})$ or an encryption of $(0^{\ell_{\mathsf{FPFE.msk}_\lambda}}, 0^{\ell_{\mathsf{PRF}.K_\lambda}}, 1, \mathsf{Sym}.k)$. $\mathcal{B}$ then sends $\mathsf{MPK} = \mathsf{FE.mpk}$ to $\mathcal{A}$. For each function query f_j that $\mathcal{A}$ sends to $\mathcal{B}$, $\mathcal{B}$ does the following: $\mathcal{B}$ computes (s_j, c_j) as in $\mathbf{Hybrid}_2^{\mathcal{A}}$. $\mathcal{B}$ then sends function query $G_j = G_{f_j, s_j, c_j}$ to its FE challenger and receives a function key $\mathsf{FE.sk}_{G_j}$ in return. This is a valid function query since for all j,

$$G_{f_j,s_j,c_j}(\mathsf{FPFE.msk}, \mathsf{PRF}.K, 0, 0^{\ell_{\mathsf{Sym}.k_\lambda}}) = G_{f_j,s_j,c_j}(0^{\ell_{\mathsf{FPFE.msk}_\lambda}}, 0^{\ell_{\mathsf{PRF}.K_\lambda}}, 1, \mathsf{Sym}.k)$$

because c_j encrypts $(\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j)$ which are generated in the same way as in the $\alpha = 0$ branch of G_{f_j,s_j,c_j}. $\mathcal{B}$ then sends $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to $\mathcal{A}$. After $\mathcal{A}$ is done making function queries, $\mathcal{A}$ outputs challenge messages $(x^{(0)}, x^{(1)})$. $\mathcal{B}$ samples $b \leftarrow \{0, 1\}$, computes CT as in $\mathbf{Hybrid}_2^{\mathcal{A}}$, sends CT to $\mathcal{A}$, and receives b' from $\mathcal{A}$. $\mathcal{B}$ outputs 1 if $b = b'$ and $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried, and outputs 0 otherwise. Observe that if $\mathsf{FE.ct}$ is an encryption of $(\mathsf{FPFE.msk}, \mathsf{PRF}.K, 0, 0^{\ell_{\mathsf{Sym}.k_\lambda}})$, then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}_2^{\mathcal{A}}$, and if $\mathsf{FE.ct}$ is an encryption of $(0^{\ell_{\mathsf{FPFE.msk}_\lambda}}, 0^{\ell_{\mathsf{PRF}.K_\lambda}}, 1, \mathsf{Sym}.k)$ then $\mathcal{B}$ emulates $\mathbf{Hybrid}_3^{\mathcal{A}}$. Additionally, $\mathcal{B}$ does not need to know $\mathsf{FE.msk}$ to carry out this experiment. Thus, by Equation 10, this means that $\mathcal{B}$ breaks the selective-IND-security of FE as $\mathcal{B}$ can distinguish between the two ciphertexts with non-negligible probability. $\square$

Hybrid$_4^{\mathcal{A}}$: We exchange the randomness generated by $\mathsf{PRF}.K$ with true randomness.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs a function size $1^{\ell_\mathcal{F}}$, a state size $1^{\ell_\mathcal{S}}$, an input size $1^{\ell_\mathcal{X}}$, and an output size $1^{\ell_\mathcal{Y}}$.

2. **Setup:**

   (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$

   (b) ~~$\mathsf{PRF}.K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$~~

   (c) $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$

   (d) $\mathsf{Sym}.k \leftarrow \mathsf{Sym.Setup}(1^\lambda)$

   (e) $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (0^{\ell_{\mathsf{FPFE.msk}_\lambda}}, 0^{\ell_{\mathsf{PRF}.k_\lambda}}, 1, \mathsf{Sym}.k))$

3. **Public Key:** Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.

4. **Function Queries:** For the j^{th} function query $f_j \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$ made by the adversary:

   (a) $s_j \leftarrow \{0,1\}^\lambda$

   (b) **Compute c_j:**

      i. ~~$(r_{\mathsf{Setup},j}, r_{\mathsf{KeyGen},j}, r_{\mathsf{EncSetup}_j} r_{\mathsf{PRF2},j}, r_{\mathsf{Enc},j}) \leftarrow \mathsf{PRF.Eval}(\mathsf{PRF}.K, s_j)$~~

      ii. $\mathsf{One\text{-}sFE.msk}_j \leftarrow \mathsf{One\text{-}sFE.Setup}(1^\lambda)$

      iii. $\mathsf{One\text{-}sFE.Enc.st}_j \leftarrow \mathsf{One\text{-}sFE.EncSetup}(\mathsf{One\text{-}sFE.msk}_j)$

      iv. $\mathsf{One\text{-}sFE.sk}_{f_j} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_j, f_j)$

      v. $\mathsf{PRF2}.k_j \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$

      vi. $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0))$

      vii. $c_j \leftarrow \mathsf{Sym.Enc}(\mathsf{Sym}.k, (\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j))$

   (c) Let $G_j = G_{f_j, s_j, c_j}$ as defined in Figure 9.

   (d) $\mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_j)$

   (e) Send $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to the adversary.

5. **Challenge Message:** $\mathcal{A}$ outputs a challenge message pair $(x^{(0)}, x^{(1)})$ where $x^{(0)} = x_1^{(0)} \dots x_n^{(0)}$ and $x^{(1)} = x_1^{(1)} \dots x_n^{(1)}$ for some length $n \in \mathbb{N}$ chosen by the adversary and where each $x_i^{(0)}, x_i^{(1)} \in \{0,1\}^{\ell_\mathcal{X}}$.

6. **Challenge Bit:** $b \leftarrow \{0,1\}$

7. **Challenge Ciphertext:**

   (a) For $i \in [n]$,

      i. $t_i \leftarrow \{0,1\}^\lambda$

      ii. Let $H_i = H_{i, x_i^{(b)}, t_i}$ as defined in Figure 8.

      iii. $\mathsf{FPFE.sk}_{H_i} = \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H_i)$

      iv. If $i = 1$, let $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})$. Else, let $\mathsf{CT}_i = \mathsf{FPFE.sk}_{H_i}$

   (b) Send $\mathsf{CT} = \{\mathsf{CT}_i\}_{i \in [n]}$ to the adversary.

8. **Experiment Outcome:** The adversary outputs a bit b'. Output 1 if $b = b'$ and $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried, and output 0 otherwise.

Lemma D.4. *If* PRF *is a secure PRF, then for all PPT adversaries* $\mathcal{A}$*,*

$$\left|\Pr[\mathbf{Hybrid}_3^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_4^{\mathcal{A}}(1^\lambda) = 1]\right| \le \mathsf{negl}(\lambda)$$

Proof. Suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$ such that

$$\left|\Pr[\mathbf{Hybrid}_3^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_4^{\mathcal{A}}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{11}$$

We build a PPT adversary $\mathcal{B}$ that breaks the security of PRF. $\mathcal{B}$ first runs $\mathcal{A}$ on input 1^λ and receives parameters $1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{S}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}}$. $\mathcal{B}$ then sends input size 1^λ, and output size $1^{5\lambda}$ to its PRF challenger. $\mathcal{B}$ is then given oracle access to either $\mathsf{PRF.Eval}(\mathsf{PRF}.K, \cdot)$ for some $\mathsf{PRF}.K \leftarrow \mathsf{PRF.Setup}(1^\lambda, 1^\lambda, 1^{5\lambda})$ or to a uniformly random function $R \leftarrow \mathcal{R}_{\lambda,5\lambda}$ where $\mathcal{R}_{\lambda,5\lambda}$ is the set of all functions from $\{0,1\}^\lambda$ to $\{0,1\}^{5\lambda}$. $\mathcal{B}$ computes $(\mathsf{FE.mpk}, \mathsf{FE.msk}, \mathsf{FPFE.msk}, \mathsf{Sym}.k)$ as in $\mathbf{Hybrid}_3^{\mathcal{A}}$ and computes $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, 0^{\ell_{\mathsf{FPFE.msk}_\lambda}}, 0^{\ell_{\mathsf{PRF}.k_\lambda}}, 1, \mathsf{Sym}.k)$. (This does not require knowledge of $\mathsf{PRF}.K$). $\mathcal{B}$ then sends $\mathsf{MPK} = \mathsf{FE.mpk}$ to $\mathcal{A}$. For each function query f_j that $\mathcal{A}$ sends to $\mathcal{B}$, $\mathcal{B}$ does the following: $\mathcal{B}$ samples $s_j \leftarrow \{0,1\}^\lambda$ and sets $(r_{\mathsf{Setup},j}, r_{\mathsf{KeyGen},j}, r_{\mathsf{EncSetup},j}, r_{\mathsf{PRF2},j}, r_{\mathsf{Enc},j})$ equal to the output of $\mathcal{B}$'s oracle on s_j. $\mathcal{B}$ then computes $\mathsf{One\text{-}sFE.msk}_j \leftarrow \mathsf{One\text{-}sFE.Setup}(1^\lambda; r_{\mathsf{Setup},j})$,
$\mathsf{One\text{-}sFE.Enc.st}_j \leftarrow \mathsf{One\text{-}sFE.EncSetup}(\mathsf{One\text{-}sFE.msk}_j; r_{\mathsf{EncSetup},j})$,
$\mathsf{One\text{-}sFE.sk}_{f_j} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_j, f_j; r_{\mathsf{KeyGen},j})$, $\mathsf{PRF2}.k_j \leftarrow \mathsf{PRF2.Setup}(1^\lambda; r_{\mathsf{PRF2},j})$, and $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0); r_{\mathsf{Enc},j})$ using these values as randomness. $\mathcal{B}$ computes c_j and SK_{f_j} from these values as in $\mathbf{Hybrid}_3^{\mathcal{A}}$ and sends SK_{f_j} to $\mathcal{A}$. After $\mathcal{A}$ is done making function queries, $\mathcal{A}$ outputs challenge messages $(x^{(0)}, x^{(1)})$. $\mathcal{B}$ samples $b \leftarrow \{0,1\}$, computes CT as in $\mathbf{Hybrid}_3^{\mathcal{A}}$, sends CT to $\mathcal{A}$, and receives b' from $\mathcal{A}$. $\mathcal{B}$ outputs 1 if $b = b'$ and $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried, and outputs 0 otherwise. Observe that if $\mathcal{B}$'s oracle was a uniform random function R, then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}_4^{\mathcal{A}}$, and if $\mathcal{B}$'s oracle was $\mathsf{PRF.Eval}(\mathsf{PRF}.K, \cdot)$, then $\mathcal{B}$ emulates $\mathbf{Hybrid}_3^{\mathcal{A}}$. Additionally, $\mathcal{B}$ does not need to know $\mathsf{PRF}.K$ to carry out this experiment. Thus, by Equation 11, this means that $\mathcal{B}$ breaks the security of PRF as $\mathcal{B}$ can distinguish between a random function and the PRF. □

$H^*_{i,x_i,x'_i,t_i,v_i}(\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, \mathsf{PRF2}.k, \beta)$:

- If $\beta = 0$
  1. $r_i \leftarrow \mathsf{PRF2.Eval}(\mathsf{PRF2}.k, t_i)$
  2. Output $\mathsf{One\text{-}sFE.ct}_i \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, i, x_i; r_i)$
- If $\beta = 1$
  1. $r_i \leftarrow \mathsf{PRF2.Eval}(\mathsf{PRF2}.k, t_i)$
  2. Output $\mathsf{One\text{-}sFE.ct}_i \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, i, x'_i; r_i)$
- Else, output v_i

Figure 11

Hybrid$_5^{\mathcal{A}}(1^\lambda)$: We replace each $H_{i,x_i^{(b)},t_i}$ with a new function $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_i}$ that has additional branches of computation.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs a function size $1^{\ell_{\mathcal{F}}}$, a state size $1^{\ell_{\mathcal{S}}}$, an input size $1^{\ell_{\mathcal{X}}}$, and an output size $1^{\ell_{\mathcal{Y}}}$.
2. **Setup:**
   (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$
   (b) $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$
   (c) $\mathsf{Sym}.k \leftarrow \mathsf{Sym.Setup}(1^\lambda)$
   (d) $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (0^{\ell_{\mathsf{FPFE.msk}_\lambda}}, 0^{\ell_{\mathsf{PRF}.k_\lambda}}, 1, \mathsf{Sym}.k))$
3. **Public Key:** Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.
4. **Function Queries:** For the j^{th} function query $f_j \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$ made by the adversary:
   (a) $s_j \leftarrow \{0,1\}^\lambda$
   (b) **Compute** c_j**:**
      i. $\mathsf{One\text{-}sFE.msk}_j \leftarrow \mathsf{One\text{-}sFE.Setup}(1^\lambda)$
      ii. $\mathsf{One\text{-}sFE.Enc.st}_j \leftarrow \mathsf{One\text{-}sFE.EncSetup}(\mathsf{One\text{-}sFE.msk}_j)$
      iii. $\mathsf{One\text{-}sFE.sk}_{f_j} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_j, f_j)$
      iv. $\mathsf{PRF2}.k_j \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$
      v. $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0))$
      vi. $c_j \leftarrow \mathsf{Sym.Enc}(\mathsf{Sym}.k, (\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j))$
   (c) Let $G_j = G_{f_j,s_j,c_j}$ as defined in Figure 9.
   (d) $\mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_j)$
   (e) Send $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to the adversary.
5. **Challenge Message:** $\mathcal{A}$ outputs a challenge message pair $(x^{(0)}, x^{(1)})$ where $x^{(0)} = x_1^{(0)} \dots x_n^{(0)}$ and $x^{(1)} = x_1^{(1)} \dots x_n^{(1)}$ for some length $n \in \mathbb{N}$ chosen by the adversary and where each $x_i^{(0)}, x_i^{(1)} \in \{0,1\}^{\ell_{\mathcal{X}}}$.

6. **Challenge Bit:** $b \leftarrow \{0,1\}$
7. **Challenge Ciphertext:**
   (a) For $i \in [n]$,
      i. $t_i \leftarrow \{0,1\}^\lambda$
      ii. $v_i = 0^{\ell_{\mathsf{One\text{-}sFE.ct}_\lambda}}$
      iii. Let $H_i = H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_i}$ as defined in Figure 11.
      iv. $\mathsf{FPFE.sk}_{H_i} = \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H_i)$
      v. If $i = 1$, let $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})$. Else, let $\mathsf{CT}_i = \mathsf{FPFE.sk}_{H_i}$
   (b) Send $\mathsf{CT} = \{\mathsf{CT}_i\}_{i\in[n]}$ to the adversary.
8. **Experiment Outcome:** The adversary outputs a bit b'. Output 1 if $b = b'$ and $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried, and output 0 otherwise.

Lemma D.5. *If* FPFE *is a function-private-selective-IND-secure* FE *scheme, then for all PPT adversaries $\mathcal{A}$,*

$$\left|\Pr[\mathbf{Hybrid}_4^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_5^{\mathcal{A}}(1^\lambda) = 1]\right| \le \mathsf{negl}(\lambda)$$

Proof. Suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$ such that

$$\left|\Pr[\mathbf{Hybrid}_4^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_5^{\mathcal{A}}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \quad (12)$$

We build a PPT adversary $\mathcal{B}$ that breaks the function-private-selective-IND-security of FPFE. $\mathcal{B}$ first runs $\mathcal{A}$ on input 1^λ and receives parameters $1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{S}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}}$. $\mathcal{B}$ then sends function size $1^{\ell_{H_\lambda}}$, input size $1^{\ell_{\mathsf{FPFE}.m_\lambda}}$, and output size $1^{\ell_{\mathsf{One\text{-}sFE.ct}_\lambda}}$ to its FPFE challenger where $\ell_{H_\lambda}, \ell_{\mathsf{FPFE}.m_\lambda}, \ell_{\mathsf{One\text{-}sFE.ct}_\lambda}$ are computed as described in the parameter section. $\mathcal{B}$ computes $(\mathsf{FE.mpk}, \mathsf{FE.msk}, \mathsf{Sym}.k, \mathsf{FE.ct})$ as in $\mathbf{Hybrid}_4^{\mathcal{A}}$ and sends $\mathsf{MPK} = \mathsf{FE.mpk}$ to $\mathcal{A}$. Let $q = q(\lambda)$ be the running time of $\mathcal{A}$. Observe that $q = \mathsf{poly}(\lambda)$ as $\mathcal{A}$ is polytime and that $\mathcal{A}$ outputs at most $q(\lambda)$ function queries on security parameter λ. For $j \in [q]$, $\mathcal{B}$ computes $(s_j, \mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j)$ as in $\mathbf{Hybrid}_4^{\mathcal{A}}$. (This does not require knowledge of $\mathsf{FPFE.msk}$ or f_j). $\mathcal{B}$ then sends challenge message pairs $\{((\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0), (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0))\}_{j\in[q]}$ to its FPFE challenger and receives $\{\mathsf{FPFE.ct}_j\}_{j\in[q]}$ where each $\mathsf{FPFE.ct}_j$ is an encryption of $(\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0)$. For each function query f_j that $\mathcal{A}$ sends to $\mathcal{B}$, $\mathcal{B}$ computes $\mathsf{One\text{-}sFE.sk}_{f_j} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_j, f_j)$, $c_j \leftarrow \mathsf{Sym.Enc}(\mathsf{Sym}.k, (\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j))$, and $\mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_{f_j,s_j,c_j})$, and sends $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to $\mathcal{A}$. (This is possible to compute as q is at least as large as the number of function queries that $\mathcal{A}$ makes). After $\mathcal{A}$ is done making function queries, $\mathcal{A}$ outputs challenge messages $(x^{(0)}, x^{(1)})$. $\mathcal{B}$ samples $b \leftarrow \{0,1\}$. Then, for $i \in [n]$, $\mathcal{B}$ does the following: $\mathcal{B}$ samples $t_i \leftarrow \{0,1\}^\lambda$ and sets $v_i = 0^{\ell_{\mathsf{One\text{-}sFE.ct}_\lambda}}$. $\mathcal{B}$ sends a challenge function pair $(H_{i,x_i^{(b)},t_i}, H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_i})$ to its FPFE challenger and receives an FPFE function key $\mathsf{FPFE.sk}_{H_i}$ which is either a function key for $H_{i,x_i^{(b)},t_i}$ or a function key for $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_i}$. This is a valid function query pair since for all $j \in [q]$,

$$\begin{aligned}&H_{i,x_i^{(b)},t_i}(\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0)\\ &= H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_i}(\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0)\end{aligned}$$

as the two function act the same when $\beta = 0$. If $i = 1$, $\mathcal{B}$ sets $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})$. Else, $\mathcal{B}$ sets $\mathsf{CT}_i = \mathsf{FPFE.sk}_{H_i}$. $\mathcal{B}$ sends $\mathsf{CT} = \{\mathsf{CT}_i\}_{i\in[n]}$ to $\mathcal{A}$, and receives b' from $\mathcal{A}$. $\mathcal{B}$ outputs 1 if $b = b'$

and $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried, and outputs 0 otherwise. Observe that if $\mathcal{B}$ received only ciphertexts and function keys for the first message or function of each of its challenge pairs, then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}_4^{\mathcal{A}}$, and if $\mathcal{B}$ received only ciphertexts and function keys for the second message or function of each of its challenge pairs, then $\mathcal{B}$ emulates $\mathbf{Hybrid}_5^{\mathcal{A}}$. Additionally, $\mathcal{B}$ does not need to know $\mathsf{FPFE.msk}$ to carry out this experiment. Thus, by Equation 12, this means that $\mathcal{B}$ breaks the function-private selective-IND security of FPFE as $\mathcal{B}$ can distinguish between the two security games with non-negligible probability. $\square$

Remark D.6. In this hybrid and future hybrids, if the number of functions queried is smaller than k, then before computing the challenge ciphertext, we carry out the function query step of the hybrid for a dummy function query f_k for the all zero function (but do not send SK_{f_k} to the adversary). This ensures that $r_{i,k}$ and $v_{i,k}$ are always well-defined.

Hybrid$^{\mathcal{A}}_{6,k,1}(1^\lambda)$: We replace v_i with $v_{i,k} = H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_i}(\mathsf{One\text{-}sFE.msk}_k, \mathsf{PRF2}.k_k, 0)$. This will allow us to later use the security of FPFE to change the input message in the k^{th} ciphertext $\mathsf{FPFE.ct}_k$ so that it uses the $\beta = 2$ branch of H_i.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs a function size $1^{\ell_\mathcal{F}}$, a state size $1^{\ell_\mathcal{S}}$, an input size $1^{\ell_\mathcal{X}}$, and an output size $1^{\ell_\mathcal{Y}}$.

2. **Setup:**

   (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$

   (b) $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$

   (c) $\mathsf{Sym}.k \leftarrow \mathsf{Sym.Setup}(1^\lambda)$

   (d) $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (0^{\ell_{\mathsf{FPFE.msk}_\lambda}}, 0^{\ell_{\mathsf{PRF}.k_\lambda}}, 1, \mathsf{Sym}.k))$

3. **Public Key:** Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.

4. **Function Queries:** For the j^{th} function query $f_j \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$ made by the adversary:

   (a) $s_j \leftarrow \{0,1\}^\lambda$

   (b) **Compute** c_j**:**

      i. $\mathsf{One\text{-}sFE.msk}_j \leftarrow \mathsf{One\text{-}sFE.Setup}(1^\lambda)$

      ii. $\mathsf{One\text{-}sFE.Enc.st}_j \leftarrow \mathsf{One\text{-}sFE.EncSetup}(\mathsf{One\text{-}sFE.msk}_j)$

      iii. $\mathsf{One\text{-}sFE.sk}_{f_j} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_j, f_j)$

      iv. $\mathsf{PRF2}.k_j \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$

      v. If $j < k$, $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 1))$

      vi. If $j = k$, $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0))$

      vii. If $j > k$, $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0))$

      viii. $c_j \leftarrow \mathsf{Sym.Enc}(\mathsf{Sym}.k, (\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j))$

   (c) Let $G_j = G_{f_j,s_j,c_j}$ as defined in Figure 9.

   (d) $\mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_j)$

   (e) Send $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to the adversary.

5. **Challenge Message:** $\mathcal{A}$ outputs a challenge message pair $(x^{(0)}, x^{(1)})$ where $x^{(0)} = x_1^{(0)} \dots x_n^{(0)}$ and $x^{(1)} = x_1^{(1)} \dots x_n^{(1)}$ for some length $n \in \mathbb{N}$ chosen by the adversary and where each $x_i^{(0)}, x_i^{(1)} \in \{0,1\}^{\ell_\mathcal{X}}$.

6. **Challenge Bit:** $b \leftarrow \{0,1\}$

7. **Challenge Ciphertext:**

   (a) For $i \in [n]$,

      i. $t_i \leftarrow \{0,1\}^\lambda$

ii. $r_{i,k} = \mathsf{PRF2.Eval}(\mathsf{PRF2}.k_k, t_i)$

iii. $v_{i,k} \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}_k, \mathsf{One\text{-}sFE.Enc.st}_k, i, x_i^{(b)}; r_{i,k})$

iv. Let $H_i = H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}$ as defined in Figure 11.

v. $\mathsf{FPFE.sk}_{H_i} = \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H_i)$

vi. If $i = 1$, let $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})$. Else, let $\mathsf{CT}_i = \mathsf{FPFE.sk}_{H_i}$

(b) Send $\mathsf{CT} = \{\mathsf{CT}_i\}_{i\in[n]}$ to the adversary.

8. **Experiment Outcome:** The adversary outputs a bit b'. Output 1 if $b = b'$ and $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried, and output 0 otherwise.

Lemma D.7. *If* FPFE *is a function-private-selective-IND-secure* FE *scheme, then for all PPT adversaries $\mathcal{A}$,*

$$\left|\Pr[\mathbf{Hybrid}_5^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_{6,1,1}^{\mathcal{A}}(1^\lambda) = 1]\right| \le \mathsf{negl}(\lambda)$$

Proof. Suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$ such that

$$\left|\Pr[\mathbf{Hybrid}_5^{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}_{6,1,1}^{\mathcal{A}}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{13}$$

We build a PPT adversary $\mathcal{B}$ that breaks the function-private-selective-IND-security of FPFE. $\mathcal{B}$ first runs $\mathcal{A}$ on input 1^λ and receives parameters $1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}}$. $\mathcal{B}$ then sends function size $1^{\ell_{H_\lambda}}$, input size $1^{\ell_{\mathsf{FPFE}.m_\lambda}}$, and output size $1^{\ell_{\mathsf{One\text{-}sFE.ct}_\lambda}}$ to its FPFE challenger where $\ell_{H_\lambda}, \ell_{\mathsf{FPFE}.m_\lambda}, \ell_{\mathsf{One\text{-}sFE.ct}_\lambda}$ are computed as described in the parameter section. $\mathcal{B}$ computes $(\mathsf{FE.mpk}, \mathsf{FE.msk}, \mathsf{Sym}.k, \mathsf{FE.ct})$ as in $\mathbf{Hybrid}_5^{\mathcal{A}}$ and sends $\mathsf{MPK} = \mathsf{FE.mpk}$ to $\mathcal{A}$. Let $q = q(\lambda)$ be the running time of $\mathcal{A}$. Observe that $q = \mathsf{poly}(\lambda)$ as $\mathcal{A}$ is polytime and that $\mathcal{A}$ outputs at most $q(\lambda)$ function queries on security parameter λ. For $j \in [q]$, $\mathcal{B}$ computes $(s_j, \mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j)$ as in $\mathbf{Hybrid}_5^{\mathcal{A}}$. (This does not require knowledge of $\mathsf{FPFE.msk}$ or f_j). $\mathcal{B}$ then sends challenge message pairs $\{((\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0), (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0))\}_{j\in[q]}$ to its FPFE challenger and receives $\{\mathsf{FPFE.ct}_j\}_{j\in[q]}$ where each $\mathsf{FPFE.ct}_j$ is an encryption of $(\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0)$. For each function query f_j that $\mathcal{A}$ sends to $\mathcal{B}$, $\mathcal{B}$ computes $\mathsf{One\text{-}sFE.sk}_{f_j} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_j, f_j)$, $c_j \leftarrow \mathsf{Sym.Enc}(\mathsf{Sym}.k, (\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j))$, and $\mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_{f_j,s_j,c_j})$, and sends $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to $\mathcal{A}$. (This is possible to compute as q is at least as large as the number of function queries that $\mathcal{A}$ makes). After $\mathcal{A}$ is done making function queries, $\mathcal{A}$ outputs challenge messages $(x^{(0)}, x^{(1)})$. $\mathcal{B}$ samples $b \leftarrow \{0,1\}$. Then, for $i \in [n]$, $\mathcal{B}$ does the following: $\mathcal{B}$ samples $t_i \leftarrow \{0,1\}^\lambda$, sets $v_i = 0^{\ell_{\mathsf{One\text{-}sFE}.ct_\lambda}}$, sets $r_{i,1} = \mathsf{PRF2.Eval}(\mathsf{PRF2}.k_1, t_i)$, and computes $v_{i,1} \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}_1, \mathsf{One\text{-}sFE.Enc.st}_1, i, x_i^{(b)}; r_{i,1})$. $\mathcal{B}$ sends challenge function pair $(H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_i}, H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,1}})$ to its FPFE challenger and receives an FPFE function key $\mathsf{FPFE.sk}_{H_i}$ which is either a function key for $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_i}$ or a function key for $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,1}}$. This is a valid function query pair since for all $j \in [q]$,

$$\begin{aligned}&H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_i}(\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0)\\ &= H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,1}}(\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0)\end{aligned}$$

as the two function act the same when $\beta = 0$. If $i = 1$, $\mathcal{B}$ sets $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})$. Else, $\mathcal{B}$ sets $\mathsf{CT}_i = \mathsf{FPFE.sk}_{H_i}$. $\mathcal{B}$ sends $\mathsf{CT} = \{\mathsf{CT}_i\}_{i\in[n]}$ to $\mathcal{A}$, and receives b' from $\mathcal{A}$. $\mathcal{B}$ outputs 1 if $b = b'$ and $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried, and outputs 0 otherwise. Observe that if $\mathcal{B}$ received only ciphertexts and function keys for the first message or function of each of its challenge pairs, then $\mathcal{B}$ exactly emulates $\textbf{Hybrid}_5^{\mathcal{A}}$, and if $\mathcal{B}$ received only ciphertexts and function keys for the second message or function of each of its challenge pairs, then $\mathcal{B}$ emulates $\textbf{Hybrid}_{6,1,1}^{\mathcal{A}}$. Additionally, $\mathcal{B}$ does not need to know $\mathsf{FPFE.msk}$ to carry out this experiment. Thus, by Equation 13, this means that $\mathcal{B}$ breaks the function-private selective-IND security of FPFE as $\mathcal{B}$ can distinguish between the two security games with non-negligible probability. $\square$

Hybrid$^{\mathcal{A}}_{6,k,2}(1^\lambda)$: We change the message encrypted in $\mathsf{FPFE.ct}_k$ so that we use the $\beta = 2$ branch of every $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}$. This allows us to remove $\mathsf{One\text{-}sFE.msk}_k$ and $\mathsf{PRF2}.k_k$ from $\mathsf{FPFE.ct}_k$.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs a function size $1^{\ell_{\mathcal{F}}}$, a state size $1^{\ell_{\mathcal{S}}}$, an input size $1^{\ell_{\mathcal{X}}}$, and an output size $1^{\ell_{\mathcal{Y}}}$.

2. **Setup:**
   (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$
   (b) $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$
   (c) $\mathsf{Sym}.k \leftarrow \mathsf{Sym.Setup}(1^\lambda)$
   (d) $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (0^{\ell_{\mathsf{FPFE.msk}_\lambda}}, 0^{\ell_{\mathsf{PRF}.k_\lambda}}, 1, \mathsf{Sym}.k))$

3. **Public Key:** Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.

4. **Function Queries:** For the j^{th} function query $f_j \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$ made by the adversary:
   (a) $s_j \leftarrow \{0,1\}^\lambda$
   (b) **Compute c_j:**
      i. $\mathsf{One\text{-}sFE.msk}_j \leftarrow \mathsf{One\text{-}sFE.Setup}(1^\lambda)$
      ii. $\mathsf{One\text{-}sFE.Enc.st}_j \leftarrow \mathsf{One\text{-}sFE.EncSetup}(\mathsf{One\text{-}sFE.msk}_j)$
      iii. $\mathsf{One\text{-}sFE.sk}_{f_j} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_j, f_j)$
      iv. $\mathsf{PRF2}.k_j \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$
      v. If $j < k$, $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 1))$
      vi. If $j = k$, $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (0^{\ell_{\mathsf{One\text{-}sFE.msk}_\lambda}}, 0^{\ell_{\mathsf{One\text{-}sFE.Enc.st}_\lambda}}, 0^{\ell_{\mathsf{PRF2}.k_\lambda}}, 2))$
      vii. If $j > k$, $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0))$
      viii. $c_j \leftarrow \mathsf{Sym.Enc}(\mathsf{Sym}.k, (\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j))$
   (c) Let $G_j = G_{f_j,s_j,c_j}$ as defined in Figure 9.
   (d) $\mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_j)$
   (e) Send $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to the adversary.

5. **Challenge Message:** $\mathcal{A}$ outputs a challenge message pair $(x^{(0)}, x^{(1)})$ where $x^{(0)} = x_1^{(0)} \dots x_n^{(0)}$ and $x^{(1)} = x_1^{(1)} \dots x_n^{(1)}$ for some length $n \in \mathbb{N}$ chosen by the adversary and where each $x_i^{(0)}, x_i^{(1)} \in \{0,1\}^{\ell_{\mathcal{X}}}$.

6. **Challenge Bit:** $b \leftarrow \{0,1\}$

7. **Challenge Ciphertext:**
   (a) For $i \in [n]$,
      i. $t_i \leftarrow \{0,1\}^\lambda$
      ii. $r_{i,k} = \mathsf{PRF2.Eval}(\mathsf{PRF2}.k_k, t_i)$
      iii. $v_{i,k} \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}_k, \mathsf{One\text{-}sFE.Enc.st}_k, i, x_i^{(b)}; r_{i,k})$
      iv. Let $H_i = H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}$ as defined in Figure 11.
      v. $\mathsf{FPFE.sk}_{H_i} = \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H_i)$
      vi. If $i = 1$, let $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})$. Else, let $\mathsf{CT}_i = \mathsf{FPFE.sk}_{H_i}$

(b) Send $\mathsf{CT} = \{\mathsf{CT}_i\}_{i\in[n]}$ to the adversary.

8. **Experiment Outcome:** The adversary outputs a bit b'. Output 1 if $b = b'$ and $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried, and output 0 otherwise.

Lemma D.8. *If* FPFE *is a function-private-selective-IND-secure* FE *scheme, then for all PPT adversaries* $\mathcal{A}$ *and for all* $k \in \mathbb{N}$,

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,1}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,2}(1^\lambda) = 1]\right| \le \mathsf{negl}(\lambda)$$

Proof. Suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$ and $k \in \mathbb{N}$ such that

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,1}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,2}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{14}$$

We build a PPT adversary $\mathcal{B}$ that breaks the function-private-selective-IND-security of FPFE. $\mathcal{B}$ first runs $\mathcal{A}$ on input 1^λ and receives parameters $1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}}$. $\mathcal{B}$ then sends function size $1^{\ell_{H_\lambda}}$, input size $1^{\ell_{\mathsf{FPFE}.m_\lambda}}$, and output size $1^{\ell_{\mathsf{One\text{-}sFE.ct}_\lambda}}$ to its FPFE challenger where $\ell_{H_\lambda}, \ell_{\mathsf{FPFE}.m_\lambda}, \ell_{\mathsf{One\text{-}sFE.ct}_\lambda}$ are computed as described in the parameter section. $\mathcal{B}$ computes $(\mathsf{FE.mpk}, \mathsf{FE.msk}, \mathsf{Sym}.k, \mathsf{FE.ct})$ as in $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,1}$ and sends $\mathsf{MPK} = \mathsf{FE.mpk}$ to $\mathcal{A}$. Let $q = q(\lambda)$ be the running time of $\mathcal{A}$. Observe that $q = \mathsf{poly}(\lambda)$ as $\mathcal{A}$ is polytime and that $\mathcal{A}$ outputs at most $q(\lambda)$ function queries on security parameter λ. For $j \in [q]$, $\mathcal{B}$ does the following: $\mathcal{B}$ computes $(s_j, \mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j)$ as in $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,1}$. (This does not require knowledge of $\mathsf{FPFE.msk}$ or f_j).

- If $j < k$, $\mathcal{B}$ sets its j^{th} challenge message pair to be $((\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 1), (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 1))$.
- If $j = k$, $\mathcal{B}$ sets its j^{th} challenge message pair to be $((\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0), (0^{\ell_{\mathsf{One\text{-}sFE.msk}_\lambda}}, 0^{\ell_{\mathsf{One\text{-}sFE.Enc.st}_\lambda}}, 0^{\ell_{\mathsf{PRF2}.k_\lambda}}, 2))$
- If $j > k$, $\mathcal{B}$ sets its j^{th} challenge message pair to be $((\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0), (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0))$

$\mathcal{B}$ then sends all q challenge message pairs to its FPFE challenger and receives $\{\mathsf{FPFE.ct}_j\}_{j\in[q]}$ where either each $\mathsf{FPFE.ct}_j$ is an encryption of the first message of the j^{th} challenge message pair, or each $\mathsf{FPFE.ct}_j$ is an encryption of the second message of the j^{th} challenge message pair. For each function query f_j that $\mathcal{A}$ sends to $\mathcal{B}$, $\mathcal{B}$ computes $\mathsf{One\text{-}sFE.sk}_{f_j} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_j, f_j), c_j \leftarrow \mathsf{Sym.Enc}(\mathsf{Sym}.k, (\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j))$, and $\mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_{f_j,s_j,c_j})$, and sends $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to $\mathcal{A}$. (This is possible to compute as q is at least as large as the number of function queries that $\mathcal{A}$ makes.) After $\mathcal{A}$ is done making function queries, $\mathcal{A}$ outputs challenge messages $(x^{(0)}, x^{(1)})$. $\mathcal{B}$ samples $b \leftarrow \{0,1\}$. Then, for $i \in [n]$, $\mathcal{B}$ does the following: $\mathcal{B}$ samples $t_i \leftarrow \{0,1\}^\lambda$, sets $r_{i,k} = \mathsf{PRF2.Eval}(\mathsf{PRF2}.k_k, t_i)$, and computes
$v_{i,k} \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}_k, \mathsf{One\text{-}sFE.Enc.st}_k, i, x_i^{(b)}; r_{i,k})$. $\mathcal{B}$ sends challenge function pair $(H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}, H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}})$ to its FPFE challenger and receives a FPFE function key $\mathsf{FPFE.sk}_{H_i}$ which is a function key for $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}$. This is a valid function query pair since for all $j \in [q]$ and $\beta \in \{0,1\}$, we clearly have,

$$\begin{aligned}
&H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}(\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, \beta) \\
&= H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}(\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, \beta)
\end{aligned}$$

and additionally,

$$\begin{aligned}&H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}(\mathsf{One\text{-}sFE.msk}_k, \mathsf{One\text{-}sFE.Enc.st}_k, \mathsf{PRF2}.k_k, 0)\\&= H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}(0^{\ell_{\mathsf{One\text{-}sFE.msk}_\lambda}}, 0^{\ell_{\mathsf{One\text{-}sFE.Enc.st}_\lambda}}, 0^{\ell_{\mathsf{PRF2}.k_\lambda}}, 2)\end{aligned}$$

as when $\beta = 2$, the output is $v_{i,k}$ which has been programmed to be equal to $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}(\mathsf{One\text{-}sFE.msk}_k, \mathsf{One\text{-}sFE.Enc.st}_k, \mathsf{PRF2}.k_k, 0)$. If $i = 1$, $\mathcal{B}$ sets $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})$. Else, $\mathcal{B}$ sets $\mathsf{CT}_i = \mathsf{FPFE.sk}_{H_i}$. $\mathcal{B}$ sends $\mathsf{CT} = \{\mathsf{CT}_i\}_{i\in[n]}$ to $\mathcal{A}$, and receives b' from $\mathcal{A}$. $\mathcal{B}$ outputs 1 if $b = b'$ and $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried, and outputs 0 otherwise. Observe that if $\mathcal{B}$ received only ciphertexts and function keys for the first message or function of each of its challenge pairs, then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,1}$, and if $\mathcal{B}$ received only ciphertexts and function keys for the second message or function of each of its challenge pairs, then $\mathcal{B}$ emulates $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,2}$. Additionally, $\mathcal{B}$ does not need to know $\mathsf{FPFE.msk}$ to carry out this experiment. Thus, by Equation 14, this means that $\mathcal{B}$ breaks the function-private-selective-IND-security of FPFE as $\mathcal{B}$ can distinguish between the two security games with non-negligible probability. □

Hybrid$^{\mathcal{A}}_{6,k,3}(1^\lambda)$: For each i, instead of sampling $r_{i,k}$ using $\mathsf{PRF2}.k_k$, we sample $r_{i,k}$ uniformly at random.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs a function size $1^{\ell_\mathcal{F}}$, a state size $1^{\ell_\mathcal{S}}$, an input size $1^{\ell_\mathcal{X}}$, and an output size $1^{\ell_\mathcal{Y}}$.
2. **Setup:**
   (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$
   (b) $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$
   (c) $\mathsf{Sym}.k \leftarrow \mathsf{Sym.Setup}(1^\lambda)$
   (d) $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (0^{\ell_{\mathsf{FPFE.msk}_\lambda}}, 0^{\ell_{\mathsf{PRF}.k_\lambda}}, 1, \mathsf{Sym}.k))$
3. **Public Key:** Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.
4. **Function Queries:** For the j^{th} function query $f_j \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$ made by the adversary:
   (a) $s_j \leftarrow \{0,1\}^\lambda$
   (b) **Compute** c_j**:**
      i. $\mathsf{One\text{-}sFE.msk}_j \leftarrow \mathsf{One\text{-}sFE.Setup}(1^\lambda)$
      ii. $\mathsf{One\text{-}sFE.Enc.st}_j \leftarrow \mathsf{One\text{-}sFE.EncSetup}(\mathsf{One\text{-}sFE.msk}_j)$
      iii. $\mathsf{One\text{-}sFE.sk}_{f_j} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_j, f_j)$
      iv. If $j \neq k$, $\mathsf{PRF2}.k_j \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$
      v. If $j < k$, $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 1))$
      vi. If $j = k$, $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (0^{\ell_{\mathsf{One\text{-}sFE.msk}_\lambda}}, 0^{\ell_{\mathsf{One\text{-}sFE.Enc.st}_\lambda}} 0^{\ell_{\mathsf{PRF2}.k_\lambda}}, 2))$
      vii. If $j > k$, $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0))$
      viii. $c_j \leftarrow \mathsf{Sym.Enc}(\mathsf{Sym}.k, (\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j))$
   (c) Let $G_j = G_{f_j, s_j, c_j}$ as defined in Figure 9.
   (d) $\mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_j)$
   (e) Send $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to the adversary.
5. **Challenge Message:** $\mathcal{A}$ outputs a challenge message pair $(x^{(0)}, x^{(1)})$ where $x^{(0)} = x_1^{(0)} \dots x_n^{(0)}$ and $x^{(1)} = x_1^{(1)} \dots x_n^{(1)}$ for some length $n \in \mathbb{N}$ chosen by the adversary and where each $x_i^{(0)}, x_i^{(1)} \in \{0,1\}^{\ell_\mathcal{X}}$.
6. **Challenge Bit:** $b \leftarrow \{0,1\}$
7. **Challenge Ciphertext:**
   (a) For $i \in [n]$,
      i. $t_i \leftarrow \{0,1\}^\lambda$
      ii. ~~$r_{i,k} \leftarrow \mathsf{PRF2.Eval}(\mathsf{PRF2}.k_k, t_i)$~~
      iii. $v_{i,k} \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}_k, \mathsf{One\text{-}sFE.Enc.st}_k, i, x_i^{(b)})$
      iv. Let $H_i = H^*_{i, x_i^{(b)}, x_i^{(0)}, t_i, v_{i,k}}$ as defined in Figure 11.
      v. $\mathsf{FPFE.sk}_{H_i} = \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H_i)$
      vi. If $i = 1$, let $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})$. Else, let $\mathsf{CT}_i = \mathsf{FPFE.sk}_{H_i}$

(b) Send $\mathsf{CT} = \{\mathsf{CT}_i\}_{i\in[n]}$ to the adversary.

8. **Experiment Outcome:** The adversary outputs a bit b'. Output 1 if $b = b'$ and $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried, and output 0 otherwise.

Lemma D.9. *If* PRF2 *is a secure PRF, then for all PPT adversaries $\mathcal{A}$ and for all $k \in \mathbb{N}$,*

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,2}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,3}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

Proof. Suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$ and $k \in \mathbb{N}$ such that

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,2}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,3}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{15}$$

We build a PPT adversary $\mathcal{B}$ that breaks the security of PRF2. $\mathcal{B}$ first runs $\mathcal{A}$ on input 1^λ and receives parameters $1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}}$. $\mathcal{B}$ then sends input size 1^λ, and output size 1^λ to its PRF2 challenger. $\mathcal{B}$ is then given oracle access to either $\mathsf{PRF2.Eval}(\mathsf{PRF2}.k_k, \cdot)$ for some $\mathsf{PRF2}.k_k \leftarrow \mathsf{PRF2.Setup}(1^\lambda, 1^\lambda, 1^\lambda)$ or to a uniformly random function $R2 \leftarrow \mathcal{R}2_{\lambda,\lambda}$ where $\mathcal{R}2_{\lambda,\lambda}$ is the set of all functions from $\{0,1\}^\lambda$ to $\{0,1\}^\lambda$. $\mathcal{B}$ computes $(\mathsf{FE.mpk}, \mathsf{FE.msk}, \mathsf{FPFE.msk}, \mathsf{Sym}.k, \mathsf{FE.ct})$ as in $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,2}$. $\mathcal{B}$ then sends $\mathsf{MPK} = \mathsf{FE.mpk}$ to $\mathcal{A}$. For each function query f_j that $\mathcal{A}$ sends to $\mathcal{B}$, $\mathcal{B}$ does the following: $\mathcal{B}$ computes $(s_j, \mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{One\text{-}sFE.sk}_{f_j})$ as in $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,2}$. If $j \neq k$, $\mathcal{B}$ also computes $\mathsf{PRF2}.k_j \leftarrow \mathsf{PRF2.Setup}(1^\lambda, 1^\lambda, 1^\lambda)$.

- If $j < k$, $\mathcal{B}$ computes $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 1))$.
- If $j = k$, $\mathcal{B}$ computes $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (0^{\ell_{\mathsf{One\text{-}sFE.msk}_\lambda}}, 0^{\ell_{\mathsf{One\text{-}sFE.Enc.st}_\lambda}}, 0^{\ell_{\mathsf{PRF2}.k_\lambda}}, 2))$.
- If $j > k$, $\mathcal{B}$ computes $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0))$.

$\mathcal{B}$ computes c_j and SK_{f_j} from these values as in $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,2}$ and sends SK_{f_j} to $\mathcal{A}$. After $\mathcal{A}$ is done making function queries, $\mathcal{A}$ outputs challenge messages $(x^{(0)}, x^{(1)})$. $\mathcal{B}$ samples $b \leftarrow \{0,1\}$. Then, for $i \in [n]$, $\mathcal{B}$ does the following: $\mathcal{B}$ samples $t_i \leftarrow \{0,1\}^\lambda$ and sets $r_{i,k}$ equal to the output of its oracle on input t_i. $\mathcal{B}$ computes $v_{i,k} \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}_k, \mathsf{One\text{-}sFE.Enc.st}_k, i, x_i^{(b)}; r_{i,k})$ and $\mathsf{FPFE.sk}_{H_i} \leftarrow \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}})$. If $i = 1$, $\mathcal{B}$ sets $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})$. Else, $\mathcal{B}$ sets $\mathsf{CT}_i = \mathsf{FPFE.sk}_{H_i}$. $\mathcal{B}$ sends $\mathsf{CT} = \{\mathsf{CT}_i\}_{i\in[n]}$ to $\mathcal{A}$, and receives b' from $\mathcal{A}$. $\mathcal{B}$ outputs 1 if $b = b'$ and $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried, and outputs 0 otherwise. Observe that if $\mathcal{B}$'s oracle was a uniform random function $R2$, then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,3}$, and if $\mathcal{B}$'s oracle was $\mathsf{PRF2.Eval}(\mathsf{PRF2}.k_k, \cdot)$, then $\mathcal{B}$ emulates $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,2}$. Additionally, $\mathcal{B}$ does not need to know $\mathsf{PRF2}.k_k$ to carry out this experiment. Thus, by Equation 15, this means that $\mathcal{B}$ breaks the security of PRF2 as $\mathcal{B}$ can distinguish between a random function and PRF2. □

Hybrid$^{\mathcal{A}}_{6,k,4}(1^\lambda)$: We now invoke the security of $\mathsf{One\text{-}sFE}$ to change $v_{i,k}$ from an encryption of $x^{(b)}$ under $\mathsf{One\text{-}sFE.msk}_k$ to an encryption of $x^{(0)}$ under $\mathsf{One\text{-}sFE.msk}_k$.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs a function size $1^{\ell_\mathcal{F}}$, a state size $1^{\ell_\mathcal{S}}$, an input size $1^{\ell_\mathcal{X}}$, and an output size $1^{\ell_\mathcal{Y}}$.

2. **Setup:**
   (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$
   (b) $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$
   (c) $\mathsf{Sym}.k \leftarrow \mathsf{Sym.Setup}(1^\lambda)$
   (d) $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (0^{\ell_{\mathsf{FPFE.msk}_\lambda}}, 0^{\ell_{\mathsf{PRF}.k_\lambda}}, 1, \mathsf{Sym}.k))$

3. **Public Key:** Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.

4. **Function Queries:** For the j^{th} function query $f_j \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$ made by the adversary:
   (a) $s_j \leftarrow \{0,1\}^\lambda$
   (b) **Compute** c_j**:**
      i. $\mathsf{One\text{-}sFE.msk}_j \leftarrow \mathsf{One\text{-}sFE.Setup}(1^\lambda)$
      ii. $\mathsf{One\text{-}sFE.Enc.st}_j \leftarrow \mathsf{One\text{-}sFE.EncSetup}(\mathsf{One\text{-}sFE.msk}_j)$
      iii. $\mathsf{One\text{-}sFE.sk}_{f_j} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_j, f_j)$
      iv. If $j \neq k$, $\mathsf{PRF2}.k_j \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$
      v. If $j < k$, $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 1))$
      vi. If $j = k$, $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (0^{\ell_{\mathsf{One\text{-}sFE.msk}_\lambda}}, 0^{\ell_{\mathsf{One\text{-}sFE.Enc.st}_\lambda}}, 0^{\ell_{\mathsf{PRF2}.k_\lambda}}, 2))$
      vii. If $j > k$, $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0))$
      viii. $c_j \leftarrow \mathsf{Sym.Enc}(\mathsf{Sym}.k, (\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j))$
   (c) Let $G_j = G_{f_j, s_j, c_j}$ as defined in Figure 9.
   (d) $\mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_j)$
   (e) Send $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to the adversary.

5. **Challenge Message:** $\mathcal{A}$ outputs a challenge message pair $(x^{(0)}, x^{(1)})$ where $x^{(0)} = x^{(0)}_1 \dots x^{(0)}_n$ and $x^{(1)} = x^{(1)}_1 \dots x^{(1)}_n$ for some length $n \in \mathbb{N}$ chosen by the adversary and where each $x^{(0)}_i, x^{(1)}_i \in \{0,1\}^{\ell_\mathcal{X}}$.

6. **Challenge Bit:** $b \leftarrow \{0,1\}$

7. **Challenge Ciphertext:**
   (a) For $i \in [n]$,
      i. $t_i \leftarrow \{0,1\}^\lambda$
      ii. $v_{i,k} \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}_k, \mathsf{One\text{-}sFE.Enc.st}_k, i, x^{(0)}_i)$
      iii. Let $H_i = H^*_{i, x^{(b)}_i, x^{(0)}_i, t_i, v_{i,k}}$ as defined in Figure 11.
      iv. $\mathsf{FPFE.sk}_{H_i} = \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H_i)$
      v. If $i = 1$, let $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})$. Else, let $\mathsf{CT}_i = \mathsf{FPFE.sk}_{H_i}$
   (b) Send $\mathsf{CT} = \{\mathsf{CT}_i\}_{i \in [n]}$ to the adversary.

8. **Experiment Outcome:** The adversary outputs a bit b'. Output 1 if $b = b'$ and $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried, and output 0 otherwise.

Lemma D.10. *If* $\mathsf{One\text{-}sFE}$ *is single-key, single-ciphertext, function-selective-IND-secure, then for all PPT adversaries $\mathcal{A}$ and for all $k \in \mathbb{N}$,*

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,3}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,4}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

Proof. Suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$ and $k \in \mathbb{N}$ such that

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,3}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,4}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{16}$$

We build a PPT adversary $\mathcal{B}$ that breaks the single-key, single-ciphertext, function-selective-IND-security of $\mathsf{One\text{-}sFE}$. $\mathcal{B}$ first runs $\mathcal{A}$ on input 1^λ and receives parameters $1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}}$. $\mathcal{B}$ then sends function size $1^{\ell_{\mathcal{F}}}$, state size $1^{\ell_{\mathcal{S}}}$, input size $1^{\ell_{\mathcal{X}}}$, and output size $1^{\ell_{\mathcal{Y}}}$ to its $\mathsf{One\text{-}sFE}$ challenger. $\mathcal{B}$ computes $(\mathsf{FE.mpk}, \mathsf{FE.msk}, \mathsf{FPFE.msk}, \mathsf{Sym}.k, \mathsf{FE.ct})$ as in $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,3}$. $\mathcal{B}$ then sends $\mathsf{MPK} = \mathsf{FE.mpk}$ to $\mathcal{A}$. For each function query f_j that $\mathcal{A}$ sends to $\mathcal{B}$, $\mathcal{B}$ does the following: $\mathcal{B}$ computes $s_j \leftarrow \{0,1\}^\lambda$. If $j \neq k$, $\mathcal{B}$ computes $\mathsf{One\text{-}sFE.msk}_j \leftarrow \mathsf{One\text{-}sFE.Setup}(1^\lambda)$, $\mathsf{One\text{-}sFE.EncSetup}(\mathsf{One\text{-}sFE.msk}_j)$, $\mathsf{One\text{-}sFE.sk}_{f_j} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_j, f_j)$, and $\mathsf{PRF2}.k_j \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$. If $j = k$, $\mathcal{B}$ sends f_k to its $\mathsf{One\text{-}sFE}$ challenger and receives a function key $\mathsf{One\text{-}sFE.sk}_{f_k}$ in return.

- If $j < k$, $\mathcal{B}$ computes $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 1))$.
- If $j = k$, $\mathcal{B}$ computes $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (0^{\ell_{\mathsf{One\text{-}sFE.msk}_\lambda}}, 0^{\ell_{\mathsf{One\text{-}sFE.Enc.st}_\lambda}}, 0^{\ell_{\mathsf{PRF2}.k_\lambda}}, 2))$.
- If $j > k$, $\mathcal{B}$ computes $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0))$.

$\mathcal{B}$ computes c_j and SK_{f_j} from these values as in $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,3}$ and sends SK_{f_j} to $\mathcal{A}$. After $\mathcal{A}$ is done making function queries, $\mathcal{A}$ outputs a challenge message pair $(x^{(0)}, x^{(1)})$. $\mathcal{B}$ samples $b \leftarrow \{0,1\}$, sends challenge message pair $(x^{(b)}, x^{(0)})$ to its $\mathsf{One\text{-}sFE}$ challenger, and receives a ciphertext $\mathsf{One\text{-}sFE.ct} = \{\mathsf{One\text{-}sFE.ct}_i\}_{i\in[n]}$ of either $x^{(b)}$ or $x^{(0)}$. Observe that if $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried by $\mathcal{A}$,[19] then this is a valid challenge message pair as for any $b \in \{0,1\}$,

$$f_k(x^{(b)}) = f_k(x^{(0)})$$

Then, for $i \in [n]$, $\mathcal{B}$ does the following: $\mathcal{B}$ samples $t_i \leftarrow \{0,1\}^\lambda$, sets $v_{i,k} = \mathsf{One\text{-}sFE.ct}_i$, and computes $\mathsf{FPFE.sk}_{H_i} \leftarrow \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}})$. If $i = 1$, $\mathcal{B}$ sets $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})$. Else, $\mathcal{B}$ sets $\mathsf{CT}_i = \mathsf{FPFE.sk}_{H_i}$. $\mathcal{B}$ sends $\mathsf{CT} = \{\mathsf{CT}_i\}_{i\in[n]}$ to $\mathcal{A}$, and receives b' from $\mathcal{A}$. $\mathcal{B}$ outputs 1 if $b = b'$ and $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried, and outputs 0 otherwise. Observe that if $\mathsf{One\text{-}sFE.ct}$ is an encryption of $x^{(b)}$, then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,3}$, and if $\mathsf{One\text{-}sFE.ct}$ is an encryption of $x^{(0)}$, then $\mathcal{B}$ emulates $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,4}$. Additionally, $\mathcal{B}$ does not need to know $\mathsf{One\text{-}sFE.msk}_k$ to carry out this experiment. Thus, by Equation 16, this means that $\mathcal{B}$ breaks the single-key, single-ciphertext, function-selective-IND-security of $\mathsf{One\text{-}sFE}$, as $\mathcal{B}$ can distinguish between the two ciphertexts with non-negligible probability. □

[19]If $\mathcal{A}$ submits any function query f_j where $f_j(x^{(0)}) \neq f_j(x^{(1)})$ then both hybrids output 0 so the distinguishing advantage is 0. For Equation 16 to hold, it must be the case that for infinitely many λ, with non-negligible probability, $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried by $\mathcal{A}$. In this proof, we restrict ourselves to the setting where $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried by $\mathcal{A}$ since a non-negligible distinguishing advantage in this restricted setting implies a non-negligible distinguishing advantage in the general setting for infinitely many λ.

Hybrid$^{\mathcal{A}}_{6,k,5}(1^\lambda)$: We now reverse the change we made in **Hybrid**$^{\mathcal{A}}_{6,k,3}$. For each i, instead of sampling $r_{i,k}$ uniformly at random, we sample $r_{i,k}$ using $\mathsf{PRF2}.k_k$.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs a function size $1^{\ell_\mathcal{F}}$, a state size $1^{\ell_\mathcal{S}}$, an input size $1^{\ell_\mathcal{X}}$, and an output size $1^{\ell_\mathcal{Y}}$.

2. **Setup:**

   (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$

   (b) $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$

   (c) $\mathsf{Sym}.k \leftarrow \mathsf{Sym.Setup}(1^\lambda)$

   (d) $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (0^{\ell_{\mathsf{FPFE.msk}_\lambda}}, 0^{\ell_{\mathsf{PRF}.k_\lambda}}, 1, \mathsf{Sym}.k))$

3. **Public Key:** Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.

4. **Function Queries:** For the j^{th} function query $f_j \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$ made by the adversary:

   (a) $s_j \leftarrow \{0,1\}^\lambda$

   (b) **Compute** c_j**:**

      i. $\mathsf{One\text{-}sFE.msk}_j \leftarrow \mathsf{One\text{-}sFE.Setup}(1^\lambda)$

      ii. $\mathsf{One\text{-}sFE.Enc.st}_j \leftarrow \mathsf{One\text{-}sFE.EncSetup}(\mathsf{One\text{-}sFE.msk}_j)$

      iii. $\mathsf{One\text{-}sFE.sk}_{f_j} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_j, f_j)$

      iv. ~~If $j \neq k$,~~ $\mathsf{PRF2}.k_j \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$

      v. If $j < k$, $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 1))$

      vi. If $j = k$, $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (0^{\ell_{\mathsf{One\text{-}sFE.msk}_\lambda}}, 0^{\ell_{\mathsf{One\text{-}sFE.Enc.st}_\lambda}}, 0^{\ell_{\mathsf{PRF2}.k_\lambda}}, 2))$

      vii. If $j > k$, $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0))$

      viii. $c_j \leftarrow \mathsf{Sym.Enc}(\mathsf{Sym}.k, (\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j))$

   (c) Let $G_j = G_{f_j,s_j,c_j}$ as defined in Figure 9.

   (d) $\mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_j)$

   (e) Send $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to the adversary.

5. **Challenge Message:** $\mathcal{A}$ outputs a challenge message pair $(x^{(0)}, x^{(1)})$ where $x^{(0)} = x_1^{(0)} \dots x_n^{(0)}$ and $x^{(1)} = x_1^{(1)} \dots x_n^{(1)}$ for some length $n \in \mathbb{N}$ chosen by the adversary and where each $x_i^{(0)}, x_i^{(1)} \in \{0,1\}^{\ell_\mathcal{X}}$.

6. **Challenge Bit:** $b \leftarrow \{0,1\}$

7. **Challenge Ciphertext:**

   (a) For $i \in [n]$,

      i. $t_i \leftarrow \{0,1\}^\lambda$

      ii. $r_{i,k} \leftarrow \mathsf{PRF2.Eval}(\mathsf{PRF2}.k_k, t_i)$

      iii. $v_{i,k} \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}_k, \mathsf{One\text{-}sFE.Enc.st}_k, i, x_i^{(0)}; r_{i,k})$

      iv. Let $H_i = H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}$ as defined in Figure 11.

      v. $\mathsf{FPFE.sk}_{H_i} = \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H_i)$

      vi. If $i = 1$, let $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})$. Else, let $\mathsf{CT}_i = \mathsf{FPFE.sk}_{H_i}$

(b) Send $\mathsf{CT} = \{\mathsf{CT}_i\}_{i\in[n]}$ to the adversary.

8. **Experiment Outcome:** The adversary outputs a bit b'. Output 1 if $b = b'$ and $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried, and output 0 otherwise.

Lemma D.11. *If* PRF2 *is a secure PRF, then for all PPT adversaries $\mathcal{A}$ and for all $k \in \mathbb{N}$,*

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,4}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,5}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

Proof. Suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$ and $k \in \mathbb{N}$ such that

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,4}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,5}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{17}$$

We build a PPT adversary $\mathcal{B}$ that breaks the security of PRF2. $\mathcal{B}$ first runs $\mathcal{A}$ on input 1^λ and receives parameters $1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}}$. $\mathcal{B}$ then sends input size 1^λ, and output size 1^λ to its PRF2 challenger. $\mathcal{B}$ is then given oracle access to either $\mathsf{PRF2.Eval}(\mathsf{PRF2}.k_k, \cdot)$ for some $\mathsf{PRF2}.k_k \leftarrow \mathsf{PRF2.Setup}(1^\lambda, 1^\lambda, 1^\lambda)$ or to a uniformly random function $R2 \leftarrow \mathcal{R}2_{\lambda,\lambda}$ where $\mathcal{R}2_{\lambda,\lambda}$ is the set of all functions from $\{0,1\}^\lambda$ to $\{0,1\}^\lambda$. $\mathcal{B}$ computes $(\mathsf{FE.mpk}, \mathsf{FE.msk}, \mathsf{FPFE.msk}, \mathsf{Sym}.k, \mathsf{FE.ct})$ as in $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,4}$. $\mathcal{B}$ then sends $\mathsf{MPK} = \mathsf{FE.mpk}$ to $\mathcal{A}$. For each function query f_j that $\mathcal{A}$ sends to $\mathcal{B}$, $\mathcal{B}$ does the following: $\mathcal{B}$ computes $(s_j, \mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{One\text{-}sFE.sk}_{f_j})$ as in $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,4}$. If $j \neq k$, $\mathcal{B}$ computes $\mathsf{PRF2}.k_j \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$

- If $j < k$, $\mathcal{B}$ computes $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 1))$.
- If $j = k$, $\mathcal{B}$ computes $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (0^{\ell_{\mathsf{One\text{-}sFE.msk}_\lambda}}, 0^{\ell_{\mathsf{One\text{-}sFE.Enc.st}_\lambda}}, 0^{\ell_{\mathsf{PRF2}.k_\lambda}}, 2))$.
- If $j > k$, $\mathcal{B}$ computes $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0))$.

$\mathcal{B}$ computes c_j and SK_{f_j} from these values as in $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,4}$ and sends SK_{f_j} to $\mathcal{A}$. After $\mathcal{A}$ is done making function queries, $\mathcal{A}$ outputs challenge messages $(x^{(0)}, x^{(1)})$. $\mathcal{B}$ samples $b \leftarrow \{0,1\}$. Then, for $i \in [n]$, $\mathcal{B}$ does the following: $\mathcal{B}$ samples $t_i \leftarrow \{0,1\}^\lambda$ and sets $r_{i,k}$ equal to the output of its oracle on input t_i. $\mathcal{B}$ computes $v_{i,k} \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}_k, \mathsf{One\text{-}sFE.Enc.st}_k, i, x_i^{(0)}; r_{i,k})$ and computes $\mathsf{FPFE.sk}_{H_i} \leftarrow \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}})$. If $i = 1$, $\mathcal{B}$ sets $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})$. Else, $\mathcal{B}$ sets $\mathsf{CT}_i = \mathsf{FPFE.sk}_{H_i}$. $\mathcal{B}$ sends $\mathsf{CT} = \{\mathsf{CT}_i\}_{i\in[n]}$ to $\mathcal{A}$, and receives b' from $\mathcal{A}$. $\mathcal{B}$ outputs 1 if $b = b'$ and $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried, and outputs 0 otherwise. Observe that if $\mathcal{B}$'s oracle was a uniform random function $R2$, then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,4}$, and if $\mathcal{B}$'s oracle was $\mathsf{PRF2.Eval}(\mathsf{PRF2}.k_k, \cdot)$, then $\mathcal{B}$ emulates $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,5}$. Additionally, $\mathcal{B}$ does not need to know $\mathsf{PRF2}.k_k$ to carry out this experiment. Thus, by Equation 17, this means that $\mathcal{B}$ breaks the security of PRF2 as $\mathcal{B}$ can distinguish betwewen a random function and PRF2. □

Hybrid$_{6,k,6}^{\mathcal{A}}(1^\lambda)$: We change the message encrypted in $\mathsf{FPFE.ct}_k$ so that it uses the $\beta = 1$ branch of every $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}$.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs a function size $1^{\ell_{\mathcal{F}}}$, a state size $1^{\ell_{\mathcal{S}}}$, an input size $1^{\ell_{\mathcal{X}}}$, and an output size $1^{\ell_{\mathcal{Y}}}$.
2. **Setup:**
   (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$
   (b) $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$
   (c) $\mathsf{Sym}.k \leftarrow \mathsf{Sym.Setup}(1^\lambda)$
   (d) $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (0^{\ell_{\mathsf{FPFE.msk}_\lambda}}, 0^{\ell_{\mathsf{PRF}.k_\lambda}}, 1, \mathsf{Sym}.k))$
3. **Public Key:** Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.
4. **Function Queries:** For the j^{th} function query $f_j \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$ made by the adversary:
   (a) $s_j \leftarrow \{0,1\}^\lambda$
   (b) **Compute** c_j**:**
      i. $\mathsf{One\text{-}sFE.msk}_j \leftarrow \mathsf{One\text{-}sFE.Setup}(1^\lambda)$
      ii. $\mathsf{One\text{-}sFE.Enc.st}_j \leftarrow \mathsf{One\text{-}sFE.EncSetup}(\mathsf{One\text{-}sFE.msk}_j)$
      iii. $\mathsf{One\text{-}sFE.sk}_{f_j} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_j, f_j)$
      iv. $\mathsf{PRF2}.k_j \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$
      v. If $j < k$, $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 1))$
      vi. If $j = k$, $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 1))$
      vii. If $j > k$, $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0))$
      viii. $c_j \leftarrow \mathsf{Sym.Enc}(\mathsf{Sym}.k, (\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j))$
   (c) Let $G_j = G_{f_j,s_j,c_j}$ as defined in Figure 9.
   (d) $\mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_j)$
   (e) Send $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to the adversary.
5. **Challenge Message:** $\mathcal{A}$ outputs a challenge message pair $(x^{(0)}, x^{(1)})$ where $x^{(0)} = x_1^{(0)} \dots x_n^{(0)}$ and $x^{(1)} = x_1^{(1)} \dots x_n^{(1)}$ for some length $n \in \mathbb{N}$ chosen by the adversary and where each $x_i^{(0)}, x_i^{(1)} \in \{0,1\}^{\ell_{\mathcal{X}}}$.
6. **Challenge Bit:** $b \leftarrow \{0,1\}$
7. **Challenge Ciphertext:**
   (a) For $i \in [n]$,
      i. $t_i \leftarrow \{0,1\}^\lambda$
      ii. $r_{i,k} \leftarrow \mathsf{PRF2.Eval}(\mathsf{PRF2}.k_k, t_i)$
      iii. $v_{i,k} \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}_k, \mathsf{One\text{-}sFE.Enc.st}_j, i, x_i^{(0)}; r_{i,k})$
      iv. Let $H_i = H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}$ as defined in Figure 11.
      v. $\mathsf{FPFE.sk}_{H_i} = \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H_i)$
      vi. If $i = 1$, let $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})$. Else, let $\mathsf{CT}_i = \mathsf{FPFE.sk}_{H_i}$

(b) Send $\mathsf{CT} = \{\mathsf{CT}_i\}_{i\in[n]}$ to the adversary.

8. **Experiment Outcome:** The adversary outputs a bit b'. Output 1 if $b = b'$ and $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried, and output 0 otherwise.

Lemma D.12. *If* FPFE *is function-private-selective-IND-secure, then for all PPT adversaries $\mathcal{A}$ and for all $k \in \mathbb{N}$,*

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,5}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,6}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

Proof. Suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$ and $k \in \mathbb{N}$ such that

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,5}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,6}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{18}$$

We build a PPT adversary $\mathcal{B}$ that breaks the function-private-selective-IND-security of FPFE. $\mathcal{B}$ first runs $\mathcal{A}$ on input 1^λ and receives parameters $1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{S}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}}$. $\mathcal{B}$ then sends function size $1^{\ell_{H_\lambda}}$, input size $1^{\ell_{\mathsf{FPFE}.m_\lambda}}$, and output size $1^{\ell_{\mathsf{One\text{-}sFE.ct}_\lambda}}$ to its FPFE challenger where $\ell_{H_\lambda}, \ell_{\mathsf{FPFE}.m_\lambda}, \ell_{\mathsf{One\text{-}sFE.ct}_\lambda}$ are computed as described in the parameter section. $\mathcal{B}$ computes $(\mathsf{FE.mpk}, \mathsf{FE.msk}, \mathsf{Sym}.k, \mathsf{FE.ct})$ as in $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,5}$ and sends $\mathsf{MPK} = \mathsf{FE.mpk}$ to $\mathcal{A}$. Let $q = q(\lambda)$ be the running time of $\mathcal{A}$. Observe that $q = \mathsf{poly}(\lambda)$ as $\mathcal{A}$ is polytime and that $\mathcal{A}$ outputs at most $q(\lambda)$ function queries on security parameter λ. For $j \in [q]$, $\mathcal{B}$ does the following: $\mathcal{B}$ computes $(s_j, \mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j)$ as in $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,5}$. (This does not require knowledge of FPFE.msk or f_j).

- If $j < k$, $\mathcal{B}$ sets its j^{th} challenge message pair to be $((\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 1), (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 1))$.
- If $j = k$, $\mathcal{B}$ sets its j^{th} challenge message pair to be $((\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 1), (0^{\ell_{\mathsf{One\text{-}sFE.msk}_\lambda}}, 0^{\ell_{\mathsf{One\text{-}sFE.Enc.st}_\lambda}}, 0^{\ell_{\mathsf{PRF2}.k_\lambda}}, 2))$
- If $j > k$, $\mathcal{B}$ sets its j^{th} challenge message pair to be $((\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0), (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0))$

$\mathcal{B}$ then sends all q challenge message pairs to its FPFE challenger and receives $\{\mathsf{FPFE.ct}_j\}_{j\in[q]}$ where either each $\mathsf{FPFE.ct}_j$ is an encryption of the first message of the j^{th} challenge message pair, or each $\mathsf{FPFE.ct}_j$ is an encryption of the second message of the j^{th} challenge message pair. For each function query f_j that $\mathcal{A}$ sends to $\mathcal{B}$, $\mathcal{B}$ computes $\mathsf{One\text{-}sFE.sk}_{f_j} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_j, f_j)$, $c_j \leftarrow \mathsf{Sym.Enc}(\mathsf{Sym}.k, (\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j))$, and $\mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_{f_j,s_j,c_j})$, and sends $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to $\mathcal{A}$. (This is possible to compute as q is at least as large as the number of function queries that $\mathcal{A}$ makes.) After $\mathcal{A}$ is done making function queries, $\mathcal{A}$ outputs challenge messages $(x^{(0)}, x^{(1)})$. $\mathcal{B}$ samples $b \leftarrow \{0,1\}$. Then, for $i \in [n]$, $\mathcal{B}$ does the following: $\mathcal{B}$ samples $t_i \leftarrow \{0,1\}^\lambda$, sets $r_{i,k} = \mathsf{PRF2.Eval}(\mathsf{PRF2}.k_k, t_i)$, and computes $v_{i,k} \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}_k, \mathsf{One\text{-}sFE.Enc.st}_k, i, x_i^{(0)}; r_{i,k})$. $\mathcal{B}$ sends a challenge function pair $(H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}, H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}})$ to its FPFE challenger and receives an FPFE function key $\mathsf{FPFE.sk}_{H_i}$ which is a function key for $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}$. This is a valid function query pair since for all $j \in [q]$ and $\beta \in \{0,1\}$, we clearly have

$$\begin{aligned}&H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}(\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, \beta)\\ &= H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}(\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, \beta)\end{aligned}$$

and additionally

$$\begin{aligned} &H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}(\mathsf{One\text{-}sFE.msk}_k, \mathsf{One\text{-}sFE.Enc.st}_k, \mathsf{PRF2}.k_k, 1) \\ &= H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}(0^{\ell_{\mathsf{One\text{-}sFE.msk}_\lambda}}, 0^{\ell_{\mathsf{One\text{-}sFE.Enc.st}_\lambda}}, 0^{\ell_{\mathsf{PRF2}.k_\lambda}}, 2) \end{aligned}$$

as when $\beta = 2$, the output is $v_{i,k}$ which has been programmed to be equal to $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}(\mathsf{One\text{-}sFE.msk}_k, \mathsf{One\text{-}sFE.Enc.st}_k, \mathsf{PRF2}.k_k, 1)$. If $i = 1$, $\mathcal{B}$ sets $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})$. Else, $\mathcal{B}$ sets $\mathsf{CT}_i = \mathsf{FPFE.sk}_{H_i}$. $\mathcal{B}$ sends $\mathsf{CT} = \{\mathsf{CT}_i\}_{i\in[n]}$ to $\mathcal{A}$, and receives b' from $\mathcal{A}$. $\mathcal{B}$ outputs 1 if $b = b'$ and $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried, and outputs 0 otherwise. Observe that if $\mathcal{B}$ received only ciphertexts and function keys for the first message or function of each of its challenge pairs, then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,6}$, and if $\mathcal{B}$ received only ciphertexts and function keys for the second message or function of each of its challenge pairs, then $\mathcal{B}$ emulates $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,5}$. Additionally, $\mathcal{B}$ does not need to know $\mathsf{FPFE.msk}$ to carry out this experiment. Thus, by Equation 18, this means that $\mathcal{B}$ breaks the function-private-selective-IND security of FPFE as $\mathcal{B}$ can distinguish between the two security games with non-negligible probability. □

Lemma D.13. *If* FPFE *is a function-private-selective-IND-secure* FE *scheme, then for all PPT adversaries* $\mathcal{A}$ *and for all* $k \in \mathbb{N}\backslash\{1\}$,

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k-1,6}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,1}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

Proof. Suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$ and a $k \in \mathbb{N}\backslash\{1\}$ such that

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k-1,5}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,k,1}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{19}$$

We build a PPT adversary $\mathcal{B}$ that breaks the function-private-selective-IND-security of FPFE. $\mathcal{B}$ first runs $\mathcal{A}$ on input 1^λ and receives parameters $1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{S}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}}$. $\mathcal{B}$ then sends function size $1^{\ell_{H_\lambda}}$, input size $1^{\ell_{\mathsf{FPFE}.m_\lambda}}$, and output size $1^{\ell_{\mathsf{One\text{-}sFE.ct}_\lambda}}$ to its FPFE challenger where $\ell_{H_\lambda}, \ell_{\mathsf{FPFE}.m_\lambda}, \ell_{\mathsf{One\text{-}sFE.ct}_\lambda}$ are computed as described in the parameter section. $\mathcal{B}$ computes $(\mathsf{FE.mpk}, \mathsf{FE.msk}, \mathsf{Sym}.k, \mathsf{FE.ct})$ as in $\mathbf{Hybrid}^{\mathcal{A}}_{6,k-1,5}$ and sends $\mathsf{MPK} = \mathsf{FE.mpk}$ to $\mathcal{A}$. Let $q = q(\lambda)$ be the running time of $\mathcal{A}$. Observe that $q = \mathsf{poly}(\lambda)$ as $\mathcal{A}$ is polytime and that $\mathcal{A}$ outputs at most $q(\lambda)$ function queries on security parameter λ. For $j \in [q]$, $\mathcal{B}$ does the following: $\mathcal{B}$ computes $(s_j, \mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j)$ as in $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,5}$.. (This does not require knowledge of $\mathsf{FPFE.msk}$ or f_j).

- If $j < k$, $\mathcal{B}$ sets its j^{th} challenge message pair to be $((\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 1), (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 1))$.
- If $j \geq k$, $\mathcal{B}$ sets its j^{th} challenge message pair to be $((\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0), (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0))$.

$\mathcal{B}$ then sends all q challenge message pairs to its FPFE challenger and receives $\{\mathsf{FPFE.ct}_j\}_{j\in[q]}$ where for $j < k$, $\mathsf{FPFE.ct}_j$ is an encryption of $(\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 1)$, and for $j \geq k$, $\mathsf{FPFE.ct}_j$ is an encryption of $(\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 0)$. For each function query f_j that $\mathcal{A}$ sends to $\mathcal{B}$, $\mathcal{B}$ computes $\mathsf{One\text{-}sFE.sk}_{f_j} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_j, f_j)$, $c_j \leftarrow \mathsf{Sym.Enc}(\mathsf{Sym}.k, (\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j))$, and $\mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_{f_j,s_j,c_j})$, and sends $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to $\mathcal{A}$. (This is possible to compute as q is at least as large as the number of function queries that $\mathcal{A}$ makes.) After $\mathcal{A}$ is done making function queries, $\mathcal{A}$ outputs challenge message pair $(x^{(0)}, x^{(1)})$. $\mathcal{B}$ samples $b \leftarrow \{0,1\}$. Then, for $i \in [n]$, $\mathcal{B}$ does the following: $\mathcal{B}$ computes $t_i \leftarrow \{0,1\}^\lambda$,

$r_{i,k-1} = \mathsf{PRF2.Eval}(\mathsf{PRF2}.k_{k-1}, t_i)$, $v_{i,k-1} \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}_{k-1}, \mathsf{One\text{-}sFE.Enc.st}_{k-1}, i, x_i^{(0)}; r_{i,k-1})$, $r_{i,k} = \mathsf{PRF2.Eval}(\mathsf{PRF2}.k_k, t_i)$, and $v_{i,k} \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}_k, \mathsf{One\text{-}sFE.Enc.st}_k, i, x_i^{(b)}; r_{i,k})$. $\mathcal{B}$ sends a challenge function pair $(H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k-1}}, H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}})$ to its FPFE challenger and receives an FPFE function key $\mathsf{FPFE.sk}_{H_i}$ which is either a function key for $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k-1}}$ or a function key for $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}$. This is a valid function query pair since for all $j \in [q]$ and $\beta \in \{0,1\}$,

$$\begin{aligned} &H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k-1}}(\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, \beta) \\ &= H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}(\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, \beta) \end{aligned}$$

as the two functions act the same when $\beta = 0$ or $\beta = 1$. If $i = 1$, $\mathcal{B}$ sets $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})$. Else, $\mathcal{B}$ sets $\mathsf{CT}_i = \mathsf{FPFE.sk}_{H_i}$. $\mathcal{B}$ sends $\mathsf{CT} = \{\mathsf{CT}_i\}_{i\in[n]}$ to $\mathcal{A}$, and receives b' from $\mathcal{A}$. $\mathcal{B}$ outputs 1 if $b = b'$ and $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried, and outputs 0 otherwise. Observe that if $\mathcal{B}$ received only ciphertexts and function keys for the first message or function of each of its challenge pairs, then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}^{\mathcal{A}}_{6,k-1,6}$, and if $\mathcal{B}$ received only ciphertexts and function keys for the second message or function of each of its challenge pairs, then $\mathcal{B}$ emulates $\mathbf{Hybrid}^{\mathcal{A}}_{6,k,1}$. Additionally, $\mathcal{B}$ does not need to know $\mathsf{FPFE.msk}$ to carry out this experiment. Thus, by Equation 19, this means that $\mathcal{B}$ breaks the function-private selective-IND security of FPFE as $\mathcal{B}$ can distinguish between the two security games with non-negligible probability. □

Hybrid$_7^{\mathcal{A}}(1^\lambda)$: We replace each $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,k}}$ with a function $H^*_{i,x_i^{(0)},x_i^{(0)},t_i,v_i}$ which is independent of b.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs a function size $1^{\ell_\mathcal{F}}$, a state size $1^{\ell_\mathcal{S}}$, an input size $1^{\ell_\mathcal{X}}$, and an output size $1^{\ell_\mathcal{Y}}$.
2. **Setup:**
   (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$
   (b) $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$
   (c) $\mathsf{Sym}.k \leftarrow \mathsf{Sym.Setup}(1^\lambda)$
   (d) $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (0^{\ell_{\mathsf{FPFE.msk}_\lambda}}, 0^{\ell_{\mathsf{PRF}.k_\lambda}}, 1, \mathsf{Sym}.k))$
3. **Public Key:** Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.
4. **Function Queries:** For the j^{th} function query $f_j \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$ made by the adversary:
   (a) $s_j \leftarrow \{0,1\}^\lambda$
   (b) **Compute** c_j**:**
      i. $\mathsf{One\text{-}sFE.msk}_j \leftarrow \mathsf{One\text{-}sFE.Setup}(1^\lambda)$
      ii. $\mathsf{One\text{-}sFE.Enc.st}_j \leftarrow \mathsf{One\text{-}sFE.EncSetup}(\mathsf{One\text{-}sFE.msk}_j)$
      iii. $\mathsf{One\text{-}sFE.sk}_{f_j} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_j, f_j)$
      iv. $\mathsf{PRF2}.k_j \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$
      v. $\mathsf{FPFE.ct}_j \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 1))$
      vi. $c_j \leftarrow \mathsf{Sym.Enc}(\mathsf{Sym}.k, (\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j))$
   (c) Let $G_j = G_{f_j,s_j,c_j}$ as defined in Figure 9.
   (d) $\mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_j)$
   (e) Send $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to the adversary.
5. **Challenge Message:** $\mathcal{A}$ outputs a challenge message pair $(x^{(0)}, x^{(1)})$ where $x^{(0)} = x_1^{(0)} \dots x_n^{(0)}$ and $x^{(1)} = x_1^{(1)} \dots x_n^{(1)}$ for some length $n \in \mathbb{N}$ chosen by the adversary and where each $x_i^{(0)}, x_i^{(1)} \in \{0,1\}^{\ell_\mathcal{X}}$.
6. **Challenge Bit:** $b \leftarrow \{0,1\}$
7. **Challenge Ciphertext:**
   (a) For $i \in [n]$,
      i. $t_i \leftarrow \{0,1\}^\lambda$
      ii. $v_i = 0^{\ell_{\mathsf{One\text{-}sFE.ct}_\lambda}}$
      iii. Let $H_i = H^*_{i,x_i^{(0)},x_i^{(0)},t_i,v_i}$ as defined in Figure 11.
      iv. $\mathsf{FPFE.sk}_{H_i} = \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H_i)$
      v. If $i = 1$, let $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})$. Else, let $\mathsf{CT}_i = \mathsf{FPFE.sk}_{H_i}$
   (b) Send $\mathsf{CT} = \{\mathsf{CT}_i\}_{i\in[n]}$ to the adversary.
8. **Experiment Outcome:** The adversary outputs a bit b'. Output 1 if $b = b'$ and $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried, and output 0 otherwise.

Lemma D.14. *If* FPFE *is a function-private-selective-IND-secure* FE *scheme, then for all PPT adversaries* $\mathcal{A}$*,*

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,q,6}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{7}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

where $q = q(\lambda)$ *is the runtime of* $\mathcal{A}$ *on security parameter* λ*.*

Proof. First, observe that if $q(\lambda)$ is the runtime of $\mathcal{A}$, then $\mathcal{A}$ outputs at most $q(\lambda)$ function queries on security parameter λ. Thus, $\mathbf{Hybrid}^{\mathcal{A}}_{6,q,6}$ always uses the $\beta = 1$ branch when encrypting $\mathsf{FPFE.ct}_j$ as in $\mathbf{Hybrid}^{\mathcal{A}}_{7}$. Now, suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$ such that

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,q,6}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{7}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{20}$$

We build a PPT adversary $\mathcal{B}$ that breaks the function-private-selective-IND-security of FPFE. $\mathcal{B}$ first runs $\mathcal{A}$ on input 1^λ and receives parameters $1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}}$. $\mathcal{B}$ then sends function size $1^{\ell_{H_\lambda}}$, input size $1^{\ell_{\mathsf{FPFE}.m_\lambda}}$, and output size $1^{\ell_{\mathsf{One\text{-}sFE.ct}_\lambda}}$ to its FPFE challenger where $\ell_{H_\lambda}, \ell_{\mathsf{FPFE}.m_\lambda}, \ell_{\mathsf{One\text{-}sFE.ct}_\lambda}$ are computed as described in the parameter section. $\mathcal{B}$ computes $(\mathsf{FE.mpk}, \mathsf{FE.msk}, \mathsf{Sym}.k, \mathsf{FE.ct})$ as in $\mathbf{Hybrid}^{\mathcal{A}}_{6,q,6}$ and sends $\mathsf{MPK} = \mathsf{FE.mpk}$ to $\mathcal{A}$. For $j \in [q]$, $\mathcal{B}$ computes $(s_j, \mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j)$ as in $\mathbf{Hybrid}^{\mathcal{A}}_{6,q,6}$. (This does not require knowledge of $\mathsf{FPFE.msk}$ or f_j). $\mathcal{B}$ then sends challenge message pairs $\{((\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 1), (\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 1))\}_{j\in[q]}$ to its FPFE challenger and receives $\{\mathsf{FPFE.ct}_j\}_{j\in[q]}$ where each $\mathsf{FPFE.ct}_j$ is an encryption of $(\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 1)$. For each function query f_j that $\mathcal{A}$ sends to $\mathcal{B}$, $\mathcal{B}$ computes $\mathsf{One\text{-}sFE.sk}_{f_j} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_j, f_j)$, $c_j \leftarrow \mathsf{Sym.Enc}(\mathsf{Sym}.k, (\mathsf{One\text{-}sFE.sk}_{f_j}, \mathsf{FPFE.ct}_j))$, and $\mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_{f_j,s_j,c_j})$, and sends $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to $\mathcal{A}$. (This is possible to compute as q is at least as large as the number of function queries that $\mathcal{A}$ makes.) After $\mathcal{A}$ is done making function queries, $\mathcal{A}$ outputs challenge message pair $(x^{(0)}, x^{(1)})$. $\mathcal{B}$ samples $b \leftarrow \{0,1\}$. Then, for $i \in [n]$, $\mathcal{B}$ does the following: $\mathcal{B}$ computes $t_i \leftarrow \{0,1\}^\lambda$, $v_i = 0^{\ell_{\mathsf{One\text{-}sFE}.ct_\lambda}}$, $r_{i,q} \leftarrow \mathsf{PRF2.Eval}(\mathsf{PRF2}.k_q, t_i)$, and $v_{i,q} \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}_q, \mathsf{One\text{-}sFE.Enc.st}_q, i, x_i^{(0)}; r_{i,q})$. $\mathcal{B}$ sends challenge function pair $(H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,q}}, H^*_{i,x_i^{(0)},x_i^{(0)},t_i,v_i})$ to its FPFE challenger and receives an FPFE function key $\mathsf{FPFE.sk}_{H_i}$ which is either a function key for $H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,q}}$ or a function key for $H^*_{i,x_i^{(0)},x_i^{(0)},t_i,v_i}$. This is a valid function query pair since for all $j \in [q]$,

$$\begin{aligned} &H^*_{i,x_i^{(b)},x_i^{(0)},t_i,v_{i,q}}(\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 1) \\ &= H^*_{i,x_i^{(0)},x_i^{(0)},t_i,v_i}(\mathsf{One\text{-}sFE.msk}_j, \mathsf{One\text{-}sFE.Enc.st}_j, \mathsf{PRF2}.k_j, 1) \end{aligned}$$

as the two function act the same when $\beta = 1$. If $i = 1$, $\mathcal{B}$ sets $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.sk}_{H_1})$. Else, $\mathcal{B}$ sets $\mathsf{CT}_i = \mathsf{FPFE.sk}_{H_i}$. $\mathcal{B}$ sends $\mathsf{CT} = \{\mathsf{CT}_i\}_{i\in[n]}$ to $\mathcal{A}$, and receives b' from $\mathcal{A}$. $\mathcal{B}$ outputs 1 if $b = b'$ and $f_j(x^{(0)}) = f_j(x^{(1)})$ for all f_j queried, and outputs 0 otherwise. Observe that if $\mathcal{B}$ received only ciphertexts and function keys for the first message or function of each of its challenge pairs, then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}^{\mathcal{A}}_{6,q,6}$, and if $\mathcal{B}$ received only ciphertexts and function keys for the second message or function of each of its challenge pairs, then $\mathcal{B}$ emulates $\mathbf{Hybrid}^{\mathcal{A}}_{7}$. Additionally, $\mathcal{B}$ does not need to know $\mathsf{FPFE.msk}$ to carry out this experiment. Thus, by Equation 20, this means that $\mathcal{B}$ breaks the function-private-selective-IND security of FPFE as $\mathcal{B}$ can distinguish between the two security games with non-negligible probability. $\square$

Lemma D.15. *For all adversaries* $\mathcal{A}$*,*

$$\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{7}(1^\lambda) = 1] \leq \frac{1}{2}$$

Proof. The messages sent to $\mathcal{A}$ in $\mathbf{Hybrid}_7^{\mathcal{A}}$ are independent of b. Thus, the probability that $\mathcal{A}$ correctly guesses b in $\mathbf{Hybrid}_7^{\mathcal{A}}$ is $\frac{1}{2}$. The lemma then follows since the probability that $\mathbf{Hybrid}_7^{\mathcal{A}}$ outputs 1 is at most the probability that $\mathcal{A}$ correctly guesses b. ☐

Thus, our lemmas give us the following corollary:

Corollary D.16. *If*

- PRF *and* $\mathsf{PRF2}$ *are secure PRFs,*
- Sym *is a secure symmetric key encryption scheme with pseudorandom ciphertexts,*
- $\mathsf{One\text{-}sFE}$ *is single-key, single-ciphertext, function-selective-IND-secure,*
- FPFE *is function-private-selective-IND-secure,*
- *and* FE *is selective-IND-secure,*

then sFE *is semi-adaptive-function-selective-IND-secure.*

Proof. By combining the hybrid indistinguishability lemmas above, we get that for all PPT adversaries $\mathcal{A}$,

$$\left|\Pr[\mathsf{ExptGuess}_{\mathcal{A}}^{\mathsf{Semi\text{-}Ad\text{-}Func\text{-}Sel\text{-}IND}}(1^\lambda) = 1]\right| = \left|\Pr[\mathbf{Hybrid}_1^{\mathcal{A}}(1^\lambda) = 1]\right| \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

The corollary then follows immediately. ☐

Corollary D.16 then implies Theorem 6.1, since as shown in Section 6, we can instantiate the required primitives from a selective-IND-secure, public-key FE scheme for $\mathsf{P/Poly}$ and a single-key, single-ciphertext, function-selective-IND-secure, secret-key, sFE scheme for $\mathsf{P/Poly}$.