Bounded-Collusion Streaming Functional Encryption from Minimal Assumptions

Kaartik Bhushan* Alexis Korb† Amit Sahai‡

Abstract

Streaming functional encryption (sFE), recently introduced by Guan, Korb, and Sahai [Crypto 2023], is an extension of functional encryption (FE) tailored for iterative computation on dynamic data streams. Unlike in regular FE, in an sFE scheme, users can encrypt and compute on the data as soon as it becomes available and in time proportional to just the size of the newly arrived data.

As sFE implies regular FE, all known constructions of sFE and FE for P/Poly require strong cryptographic assumptions which are powerful enough to build indistinguishability obfuscation. In contrast, *bounded-collusion* FE, in which the adversary is restricted to making at most Q function queries for some polynomial Q determined at setup, can be built from the minimal assumptions of public-key encryption (for public-key FE) [Sahai and Seyalioglu, CCS 2010; Gorbunov, Vaikuntanathan, and Wee, CRYPTO 2012] and secret-key encryption (for secret-key FE)[Ananth, Vaikuntanathan, TCC 2019].

In this paper, we introduce and build bounded-collusion *streaming* FE for any polynomial bound Q from the same minimal assumptions of public-key encryption (for public-key sFE) and secret-key encryption (for secret-key sFE). Similarly to the original sFE paper of Guan, Korb, and Sahai, our scheme satisfies semi-adaptive-function-selective security which is similar to standard adaptive indistinguishability-based security except that we require all functions to be queried before any of the challenge messages.

Along the way, our work also replaces a key ingredient (called One-sFE) from the original work of Guan, Korb, and Sahai with a much simpler construction based on garbled circuits.

*IIT Bombay. Email: kaartikbhushan@gmail.com. ORCID: 0009-0006-5616-3812.
†UCLA. Email: alexiskorb@cs.ucla.edu. ORCID: 0000-0001-6888-5296.
‡UCLA. Email: sahai@cs.ucla.edu. ORCID: 0000-0003-2216-9600

Contents

1 Introduction **3**

2 Technical Overview **7**
2.1 Single-Key, Single-Ciphertext, Secret-Key Streaming FE 7
2.2 Bootstrapping to a Q-Bounded Public-Key Streaming FE 14

3 Preliminaries **19**
3.1 Functional Encryption 20
3.2 Streaming Functional Encryption 22

4 Single-Key, Single-Ciphertext, Secret-Key Streaming FE **26**
4.1 Parameters 26
4.2 Construction 27
4.3 Correctness and Efficiency 29
4.4 Security 30

5 Bootstrapping to a Q-Bounded Public-Key sFE Scheme **47**
5.1 Parameters 48
5.2 Construction 49
5.3 Correctness and Efficiency 51
5.4 Security 53

6 Acknowledgements **81**

7 References **82**

A Preliminaries Continued **86**
A.1 Standard Notions 86
A.2 Secret-Key Functional Encryption 88
A.3 Secret-Key Streaming Functional Encryption 90

1 Introduction

Streaming functional encryption (sFE) [GKS23] is an extension of functional encryption (FE) [SW05, BSW11, O'N10] designed for scenarios involving iterative computation on dynamic and evolving data streams. In a standard FE scheme, given an encryption of x generated using the master public key, and a function key for some function f generated using the master secret key, a user should be able to compute $f(x)$ and nothing else. In streaming FE, we extend the message space to data streams $x = x_1 \dots x_n$, and the function space to streaming functions which can perform iterative computation on these message streams. Furthermore, we allow encryption and decryption to be done piece by piece as the data becomes available.

In more detail, streaming FE considers streaming functions which are stateful functions that take as input a value x_i and a state st_i, and output a value y_i and the next (updated) state st_{i+1}. We say that the output of streaming function f on a stream $x = x_1 \dots x_n$ (denoted $f(x)$) is the sequence of values $y = y_1 \dots y_n$ resulting from iteratively computing $(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$ starting from $\mathsf{st}_1 = \perp$.

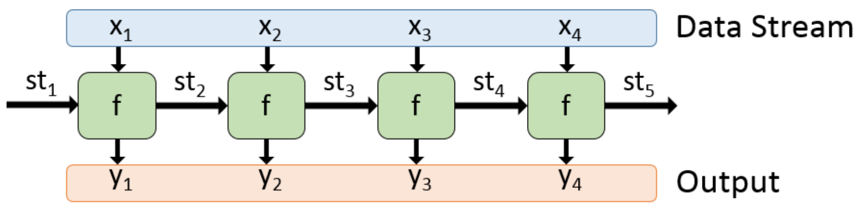


Figure 1: Computation of a streaming function f on a data stream $x = x_1 \dots x_4$.

Using the master secret key, an authority can generate function keys for streaming functions f of their choice. Then, as soon as the i^{th} value x_i of the stream becomes available, a user can generate a ciphertext ct_i for x_i using the master public key.[1] Finally, given a function key for f and access to a stream of ciphertexts $\mathsf{ct}_1 \dots \mathsf{ct}_n$ encrypting stream $x = x_1 \dots x_n$, a user can iteratively decrypt each ciphertext ct_i (as soon as it arrives) to learn the corresponding i^{th} output y_i of $f(x)$. For security, we desire that the user only learns $f(x) = y_1 \dots y_n$ and nothing else. Furthermore, we require the scheme to be *streaming efficient*, meaning that the runtime of all algorithms should be independent of the stream length n.

We remark that the standard notion of FE has garnered significant attention in the literature (e.g. [SS10, GVW12, GGH+13, GKP+13, GGG+14a, GJKS15, AV19, AMVY21, JLS21, JLS22]), with major applications like building indistinguishability obfuscation[2] (iO) [AJ15, BV15]. iO itself is very powerful and can be used to build a wide-variety of objects [SW14, CLTV15, BPR15]. FE has also been used to build other cryptographic applications, including 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].

Our goal: Building sFE from weaker/minimal assumptions. The only known constructions of sFE are [GKS23, DGKS24], the latter of which builds an adaptively secure sFE scheme

[1] Ciphertexts corresponding to the same input stream $x = x_1 \dots x_n$ must be generated under the same encryption state $\mathsf{Enc.st}$ which can be generated once using the master public key and which does not need to be updated during encryption. This is to prevent mix and match attacks between ciphertexts of different streams.

[2] The particular notion of FE that can be used to build iO is called compact FE, where the runtime of the encryption algorithm is assumed to be independent of the function sizes supported by the scheme.

from $i\mathcal{O}$ and injective PRGs, and the former which builds a semi-adaptive-function-selectively[3] secure sFE scheme from standard FE.[4] Both these constructions require the usage of heavy duty primitives. FE and $i\mathcal{O}$ are notoriously complex to build and, at present, requires three different cryptographic assumptions [JLS22]. Indeed, these difficulties seem inherent to the construction of sFE, since sFE implies standard FE which implies $i\mathcal{O}$. However, we would ideally like to build a version of sFE which does not require such strong assumptions.

This gives us the following goal: Can we construct a notion of sFE which maintains a meaningful notion of security, but that requires much weaker assumptions than $i\mathcal{O}$ or FE? To this end, we will construct a *bounded-collusion*[5] sFE scheme from the minimal assumptions of public-key encryption (for public-key sFE) and secret-key encryption (for secret-key sFE). As we explain below, this notion of security is still meaningful in many real world scenarios.

The advantages of sFE, even with bounded collusion. Streaming FE allows us to extend the usage of FE to a variety of new applications and scenarios in which using standard FE may incur a significant cost in efficiency or usability. In particular, sFE is especially useful in situations where the data we wish to encrypt either is not concurrently available, is too large to store or compute on all at once, or is being continually added to and updated. All of these situations make it difficult to compute on the data in one go, which therefore creates a need to process the data in batches or as a *stream*. Furthermore, sFE allows users to obtain partial outputs as and when the encrypted data becomes available, rather than needing to wait for all of the data to arrive.

As a motivating example, consider the following use case: Suppose several medical institutions would like to run machine-learning algorithms on a large set of patient data stored by some hospital. While the hospital is supportive of this research, it cannot simply give out the patient data as that would violate patient privacy laws. However, using sFE we can easily facilitate this process! The hospital can first generate and distribute function keys for each of the machine learning algorithms requested by the research institutions. Then, the hospital can start encrypting its large medical database in batches as the data becomes available and as computing resources are freed up. The flexibility of being able to encrypt smaller chunks of data at a time allows the hospital to both use a smaller number of simultaneous computing resources and to easily incorporate any new patient data it may gather from future patients. As soon as the first batch of encrypted data arrives, the research institutions can begin processing their algorithms on the data. Then, upon receipt of each subsequent batch of encrypted data, the institutions can update their algorithms to incorporate the new data in time proportional to the size of just the newly arrived data. The correctness of sFE ensures that each institution will learn the output of its algorithm on the data received so far, while the security of sFE ensures that no other patient data is leaked.

Observe that if we had tried to use standard FE rather than sFE, then this process would become much more difficult as both the hospital and the research instutions would need to compute on the entire database all at once. This requires both parties to have much larger simultaneous computing power and also may incur delays since they would need to wait until all of the data becomes available to them. Furthermore, if the data evolves or changes, then the entire encryption and decryption

[3]Semi-adaptive-function-selective security is similar to adaptive security, except that we require all function queries to come before any message query.

[4]In particular, [GKS23] build their scheme from a *selectively* secure FE scheme, in which security is only required to hold when the challenge messages are chosen at the beginning of the experiment. Though [GKS23] state that their scheme must also be *compact*, this is not a necessary requirement since compact, selectively secure FE for P/Poly can be built from (non-compact) selectively secure FE for P/Poly [AJS15, BV15].

[5]This is the notion where an adversary can only query for an a-priori bounded number of function keys in the security experiment.

process would need to be restarted.

Bounded collusions. In the example use case mentioned above, the hospital may need to generate a large number of ciphertexts, since each institution may wish to run their algorithms on different sets (or streams) of evolving patient data, depending on the nature of the experiment. However, the number of function keys that are ever left outstanding is equal to the number of current medical researchers working on patient data. This group may not be exorbitantly large since computing on the data requires both a large amount of computing power as well as permission (and appropriate background checks) from the hospital or regulators. Thus, it could be reasonable to place an a priori bound on the number of researchers allowed to concurrently operate on the data. Combined with careful data deletion policies for no longer needed function keys, the hospital could ensure that the number of function keys in circulation never exceeds this bound.

This brings us to the notion of bounded-collusion sFE, which is a variant of sFE in which security is only required to hold when the number of function keys released does not exceed some a-priori bound Q specified during setup. We will call Q the collusion bound, and will use the term Q-bounded to refer to a bounded-collusion scheme which is secure against Q function key queries. While weaker than the standard notion of security, this notion of security still permits many natural use cases such as the one described above.

Indeed, bounded-collusion security for standard FE and other related primitives (such as IBE and ABE) has been studied extensively in the literature (e.g. [SS10, GLW12, GVW12, AR17, Agr17, ISV$^+$17, GKW18, AV19, GSW21, Wee21, AKM$^+$22, GGLW22, GGL24, DKXY02, HK04, CHH$^+$07]). As it turns out, there is a massive gap between the assumptions needed to build fully secure FE and bounded-collusion FE. While fully secure FE requires the same assumptions needed to build $i\mathcal{O}$, Q-bounded FE for any polynomial Q of the security parameter can be built from the minimal assumptions of one-way functions (for secret-key FE) or public-key encryption (for public-key FE) [AV19]. Given this massive difference in assumptions, it is natural to ask whether a similar difference holds for sFE. Thus, we ask the following question:

> *Can we construct a bounded-collusion sFE scheme from weaker (minimal) assumptions such as one-way functions or public-key encryption?*

Our results. In this paper, we answer the question in the affirmative and prove the following theorem:

Theorem 1.1 ((Informal))**.** *Assuming the existence of CPA-secure public-key (resp. secret-key) encryption, there exists a Q-bounded, semi-adaptive-function-selectively secure, public-key (resp. secret-key)* sFE *scheme for* $\mathsf{P/Poly}$ *for any polynomial $Q = Q(\lambda)$ of the security parameter λ.*

Our assumptions are *identical* to those needed for building adaptively secure, bounded-collusion standard FE [AV19], and indeed are the minimal assumptions needed to build this primitive. (Note also that one-way functions are equivalent to CPA-secure secret-key encryption.) Additionally, this is the first sFE construction that does not depend on assumptions which imply $i\mathcal{O}$.

Our final scheme is Q-bounded, semi-adaptive-function-selectively secure (see Definition 3.15), which is the same as standard bounded-collusion indistinguishability based security except that we require all function keys to be queried before any challenge message queries. We remark that this restriction on the ordering of function and message queries is not a novel development of our work. The original construction of sFE in [GKS23] achieved only semi-adaptive-function-selective security which is the corresponding notion of security in the unbounded-collusion setting. Furthermore, while [DGKS24] - the only other known construction of sFE - achieved adaptive security, they were only able to do so by utilizing complex $i\mathcal{O}$-based techniques, which we do not wish to use here. We leave the construction of Q-bounded, *adaptively* secure sFE from weaker assumptions as an interesting open problem.

Indeed, not only do we achieve new results, we also significantly simplify the construction of [GKS23] along the way. Let's recall the basic 2-step blueprint of [GKS23]:

1. First, they construct a secret-key sFE scheme One-sFE that is only required to be secure against one challenge function and one challenge stream.

2. Then, they bootstrap this scheme into a bounded-collusion public-key (or secret-key) sFE scheme.

Our construction completely reworks the first step in a much simpler way, using minimal assumptions. This reworked step can completely replace the much more complicated FE-based construction found in [GKS23]. We also make some modifications to the second step which are needed in order to make it work in the bounded-collusion setting.

In more detail, in prior work, the first step relied upon a recursive FE computation, which led to circular parameter dependencies. To solve these parameter issues, prior work had to add a lot of strong assumptions and additional machinery, including using two alternating FE schemes, one of which was *strongly-compact*.[6] In our construction, we build One-sFE using just one way functions!

For the second step, prior work required unbounded-collusion FE. As we only know how to build such FE schemes from strong assumptions, we had to modify the procedure. We were able to downgrade the unbounded-collusion FE scheme to a bounded-collusion FE scheme by careful reorganization and restructuring of the construction.

Related works. As mentioned, [DGKS24] builds an adaptively secure sFE scheme from $i\mathcal{O}$ and injective PRGs, and [GKS23] builds a semi-adaptive-function-selectively secure sFE scheme from standard FE.

There has been a long line of work on bounded-collusion FE [SS10, GVW12, AR17, Agr17, GKW18, AV19] culminating in the construction of public-key bounded-collusion FE from PKE. [GGLW22, GGL24] construct a *dynamically*-bounded FE scheme from IBE.[7] [AMVY21, AKM+22] build bounded-collusion FE schemes for TMs.

Two two types of FE most similar to sFE are FE for Turing machines [GKP+13, AS16] and multi-input FE (MIFE) [GGG+14b, ACF+19, BKS16, GJO16]. While FE for Turing machines also involves iterative computation, unlike sFE, the entire input must be known at encryption time and no output can be generated before the computation is completed. MIFE, like sFE, allows for different portions of the input to be encrypted at different times. However, in MIFE, decryption can only occur once the decryptor receives ciphertexts for all portions of the input. In contrast, in sFE, the decryptor can begin decryption on the stream of ciphertexts as soon as they arrive.

[6] A strongly-compact FE scheme is one where the size of the setup and encryption circuits are independent of the size of the functions used in key generation.

[7] In a dynamically-bounded FE scheme, the collusion-bound Q can be independently specified for each new ciphertext. In particular, the collusion-bound does not have to be chosen at setup.

2 Technical Overview

Following the general blueprint used in prior work [GKS23,DGKS24], we will construct our bounded-collusion sFE scheme in two steps:

1. First, we construct a secret-key sFE scheme $\mathsf{One\text{-}sFE}$ that is only secure against adversaries who are given just one function key followed by one encrypted challenge stream. We prove the following:

   Theorem 2.1. *Assuming* OWF*s, there exists a single-key, single-ciphertext, function-selectively secure, secret-key sFE scheme for* $\mathsf{P/Poly}$.

2. We then bootstrap $\mathsf{One\text{-}sFE}$ into a bounded-collusion, public-key sFE scheme.

   Theorem 2.2. *Assuming*

   (a) a Q-bounded, adaptively secure, public-key (resp. secret-key) FE scheme for $\mathsf{P/Poly}$

   (b) a single-key, single-ciphertext, function-selectively secure, secret-key sFE scheme for $\mathsf{P/Poly}$

   there exists a Q-bounded, semi-adaptive-function-selectively secure, public-key (resp. secret-key) sFE scheme for $\mathsf{P/Poly}$.

As bounded-collusion FE can be built from PKE (for public-key FE) or OWFs (for secret-key FE) [AV19], this gives us our main theorem (Theorem 1.1).

Contrasting this with prior work, for Step 1, [GKS23] construct $\mathsf{One\text{-}sFE}$ from *compact*[8] FE, and [DGKS24] construct an *adaptively* secure variant of $\mathsf{One\text{-}sFE}$ using $i\mathcal{O}$ and injective PRGs. We remark that since compact FE can be used to build $i\mathcal{O}$ [BV15], we only know how to build compact FE from the same assumptions needed to build $i\mathcal{O}$. In particular, we do not have any constructions of compact FE from PKE/OWFs. For Step 2, both [GKS23] and [DGKS24] use *unbounded-collusion* FE to bootstrap their $\mathsf{One\text{-}sFE}$ scheme into an *unbounded-collusion*, public-key sFE scheme. Their final scheme maintains the same type of security (i.e. function-selective or adaptive) as their $\mathsf{One\text{-}sFE}$ scheme. In contrast, we will bootstrap our scheme using *bounded-collusion* FE. As we will explain below, this requires some non-trivial changes to the bootstrapping construction.

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

We will first focus on constructing $\mathsf{One\text{-}sFE}$. As all prior work crucially required either compact FE or $i\mathcal{O}$, which we do not know how to build from PKE/OWFs, we will need new ideas.

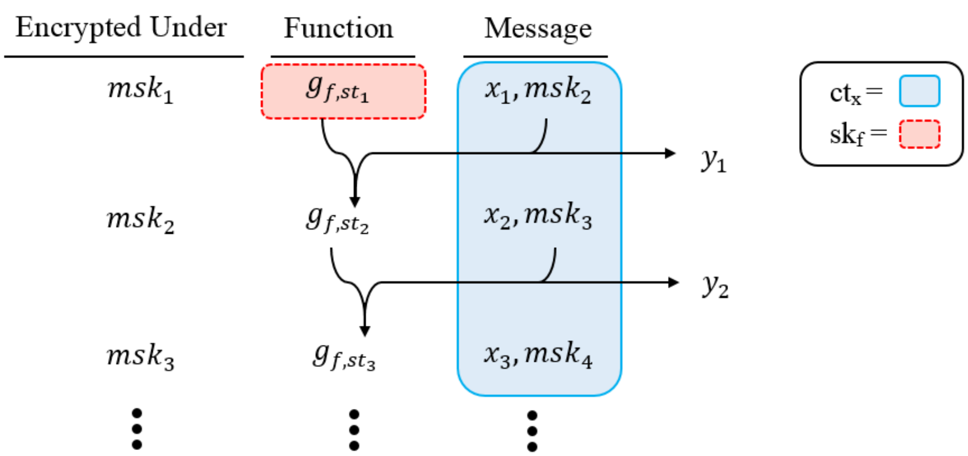
Starting Point: Iterative Use of Functional Encryption As our starting point, we consider the following natural idea: We will use regular FE to execute each iteration of our streaming function. Here, we assume the existence of a simulation secure, single-key, single-ciphertext, secret-key functional encryption scheme FE which can be built from OWFs [SS10, GVW12]. Since this scheme is only secure for one message and one key, we will use a different FE scheme for every iteration i.[9]

[8] Specifically, they require that the size of the setup and encryption algorithm of the FE scheme are independent of the size of the functions for which function keys are generated.

[9] This was also an idea from [GKS23].

Now, in a streaming FE scheme, we need to combine three elements in each iteration: a function f, an input x_i, and a state st_i. However, regular FE only allows us to securely combine two elements: a function g and a message m. Therefore, we will place both the function f and the state st_i inside the function key and will place the stream value x_i in the ciphertext. Then, our FE scheme will allow us to securely compute $(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$. In order to pass on st_{i+1} to the FE scheme of the next iteration, we will have the i^{th} FE scheme output a function key containing both f and st_{i+1} under the $(i+1)^{th}$ scheme. This gives us the following construction, depicted in Figure 2:[10]

In more detail, our master secret key will be a PRF key K, which can be used to generate FE master secret keys msk_i for each iteration i. To encrypt the i^{th} stream value x_i, we will encrypt x_i and the master secret key msk_{i+1} for the next iteration under the i^{th} FE scheme. To generate a function key for f, we will create a function key for g_{f,st_1} (where $\mathsf{st}_1 = \perp$ and g_{f,st_1} is as defined below) using the 1^{st} FE scheme. This will enable us to begin the decryption process, starting at iteration 1. We can then decrypt the ciphertext for x_i using the i^{th} FE scheme and the function key for g_{f,st_i} generated from the previous iteration (or given in the function key if $i = 1$) to get the correct output value y_i and the function key for the next iteration.



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

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

Figure 2: Initial Idea for Building One-sFE

The correctness of our One-sFE scheme follows from the correctness of the underlying FE scheme. For security, the idea is to sequentially simulate each FE scheme starting from iteration 1. Observe that simulating the i^{th} scheme will hide the values present in the i^{th} ciphertext, namely x_i and msk_{i+1}. This means that after simulating the i^{th} FE scheme, msk_{i+1} will be removed from the experiment, which will allow us to invoke the security of the $(i+1)^{th}$ FE scheme in the following iteration. Then, after simulating every iteration, the entire stream $x = x_1 \dots x_n$ will be hidden.

Unfortunately, there are a few issues with the scheme laid out above.

[10]Figure 2 is imported from [GKS23] as our starting idea is the same as the starting idea from [GKS23].

1. Standard FE only allows us to generate function keys for deterministic functionalities. However, since $\mathsf{FE.KeyGen}$ is, in general, a randomized function, then g_{f,st_i} may also be a randomized function.

2. Since standard FE does not guarantee function privacy, the intermediate states st_i, which are placed in the function keys, are not hidden. This compromises the security of our sFE scheme as we require both the stream values and the intermediate states to remain hidden.

3. Our definition of g_{f,st_i} is recursive since it needs to output a function key for $g_{f,\mathsf{st}_{i+1}}$. Since we are using an FE scheme for *circuits*, this means that the size of each g_{f,st_i} must be strictly larger than the size of $g_{f,\mathsf{st}_{i+1}}$. Thus, the size of the initial function g_{f,st_1} will depend on the total number of iteration n we wish to compute, breaking the efficiency requirements of our sFE scheme.

The first two issues end up being relatively easy to solve. For the first issue, we can either instantiate FE with a scheme that has a deterministic key generation algorithm (such as [SS10]) or can provide the randomness needed for key generation in the ciphertexts.

For the second issue, rather than placing st_i directly in the function key, we will instead place an encryption $\widetilde{\mathsf{st}}_i$ of st_i (using a one-time pad) in the function key. When encrypting x_i, we will also generate and encrypt one-time pads p_i and p_{i+1}. These pads will be generated using a PRF key K which will be the master secret key of our $\mathsf{One\text{-}sFE}$ scheme. We will then modify each function $g_{f,\widetilde{\mathsf{st}}_i}$ (as shown below) so that we encrypt and decrypt the intermediate states as appropriate using the one-time pads p_i, p_{i+1} provided by the corresponding ciphertext. Observe that since our security proof relies on us simulating every ciphertext, then we will eventually hide all of the one-time pads p_i, and thus will eventually hide all of the states st_i.

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

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 $(y_i, \mathsf{FE.KeyGen}(\mathsf{msk}_{i+1}, g_{f,\widetilde{\mathsf{st}}_{i+1}}))$.

Figure 3: Definition of $g_{f,\widetilde{\mathsf{st}}_i}$.

The third issue, however, ends up being quite problematic. In [GKS23], they solve the third issue by using a complicated construction which involves splitting their FE scheme into two alternating FE schemes and utilizing the *compactness* of one of these FE scheme to prevent circular parameter dependencies. However, as previously mentioned, we do not wish to use compact FE as we are trying to build our scheme from simple assumptions such as OWFs.

In this paper, rather than just adding machinery on top of this blueprint (as was done in [GKS23]), we will instead instantiate the underlying FE scheme with a particular construction, and then make non-black-box modifications which involve the particulars of both the choice of FE scheme and the general blueprint. Our starting FE scheme will be from [SS10] which we describe below.

Summary of [SS10]. We now summarize the simulation-secure, secret-key FE scheme from [SS10] which is only secure against adversaries who are given one function key followed by one challenge ciphertext. Let ℓ denote the length of circuits supported by our FE scheme, and let SKE be a secret-key encryption scheme.

- $\mathsf{Setup}(1^\lambda)$: Generate 2ℓ SKE keys $\{\mathsf{sk}_{j,b}\}_{j\in[\ell_{\mathcal{F}}],b\in\{0,1\}}$. Output these keys as the MSK.
- $\mathsf{Enc}(\mathsf{MSK}, x)$: Define circuit U_x which takes as input a function f and computes $U_x(g) = g(x)$. Garble U_x using a circuit garbling scheme (e.g. [Yao86]) to get garbled circuit $\widetilde{U}$ and input labels $\{\mathsf{lab}_{j,b}\}_{j\in[\ell],b\in\{0,1\}}$. Encrypt each label $\mathsf{lab}_{j,b}$ under $\mathsf{sk}_{j,b}$ to get $\mathsf{ct}_{j,b}$. Output $\mathsf{CT} = (\widetilde{U}, \{\mathsf{ct}_{j,b}\}_{j\in[\ell],b\in\{0,1\}})$.
- $\mathsf{KeyGen}(\mathsf{MSK}, g)$: Output $\mathsf{SK}_g = \{\mathsf{sk}_{j,g[j]}\}_{j\in[\ell]}$ where $g[j]$ is the j^{th} bit of g.
- $\mathsf{Dec}(\mathsf{SK}_g, \mathsf{CT}_x)$: Use the secret keys $\{\mathsf{sk}_{j,g[j]}\}_{j\in[\ell]}$ from SK_g to decrypt the corresponding ciphertexts from CT_m and recover $\{\mathsf{lab}_{j,g[j]}\}_{j\in[\ell]}$. Then, evaluate the garbled circuit $\widetilde{U}$ on these labels to learn $U_x(g) = g(x)$.

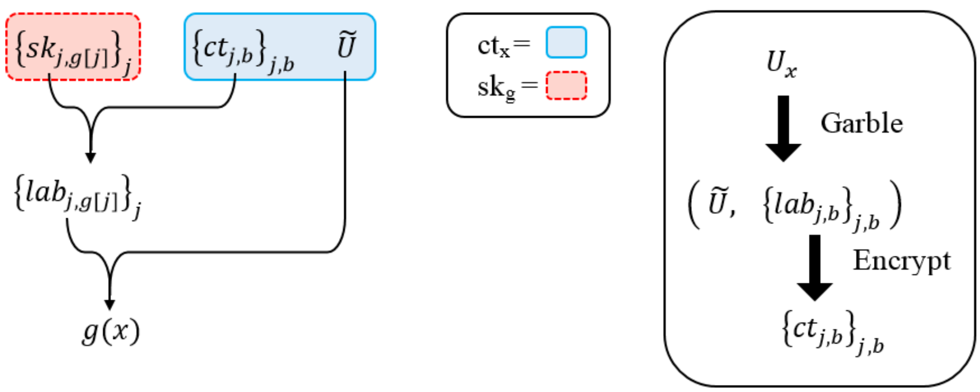


Figure 4: FE scheme from [SS10].

Correctness follows from the correctness of SKE and the garbling scheme. To argue security, we can first switch all ciphertexts $\mathsf{ct}_{j,1-g[j]}$ for the unused labels $\mathsf{lab}_{j,1-g[j]}$ to encryptions of $\perp$ since the corresponding secret keys $\mathsf{sk}_{j,1-g[j]}$ are kept hidden from the adversary. We can then use the security of the garbling scheme to simulate both the garbled circuit $\widetilde{U}$ for x and the input labels $\{\mathsf{lab}_{j,g[j]}\}_{j\in[\ell_{\mathcal{F}}]}$ for g from $(g, g(x) = U_x(g))$. Since the circuit $\widetilde{U}$ is now being simulated, nothing about the input x is leaked beyond what is revealed by $g(x)$, giving us the desired security.

Can we just use [SS10] directly? Suppose we directly plug [SS10] into our initial construction (Figure 2). Then, the ciphertext for x_i will consist of a garbled univeral circuit U_i (with x_i and other values hardwired into it) and encryptions of the corresponding input labels. The function key for iteration i will simply be a series of secret keys, one for each bit in the description of $g_{f,\widetilde{\mathsf{st}}_i}$. Unfortunately, as previously mentioned, we would still have an efficiency/size problem with our function keys since the definition of $g_{f,\widetilde{\mathsf{st}}_i}$ is recursive.

Exploiting the structure of garbled circuits. However, we observe that in the construction of [SS10], the structure of the function key is relatively simple, and most of the heavy lifting is done by the ciphertext. In particular, the function key for g_{f,st_i} depends only on the bit-string description of g_{f,st_i}.

Our key observation is that the description of our function g does not change much between iterations. In particular, the only thing that changes in the description of g between iteration i and $i+1$ is that the encrypted state changes from $\widetilde{\mathsf{st}}_i$ to state $\widetilde{\mathsf{st}}_{i+1}$. Thus, the only parts of the description of g_{f,st_i} that are unknown to the encryptor at encryption time are the function f and the encrypted state st_i. Therefore, we can greatly simplify our function keys by offloading the static parts of g to the encryptor! As we will show, this change will solve the issue of exploding key sizes!

Let us go into more detail. If we we directly plug in [SS10] to our initial construction, then the ciphertext for x_i consists of a garbled circuit and encrypted input labels for the following function:

> $U[x_i, p_i, p_{i+1}, \mathsf{msk}_{i+1}](g_{f,\widetilde{\mathsf{st}}_i})$:
>
> 1. Output $g_{f,\widetilde{\mathsf{st}}_i}(x_i, p_i, p_{i+1}, \mathsf{msk}_{i+1})$

where we define $g_{f,\widetilde{\mathsf{st}}_i}$ as in Figure 3 and where msk_{i+1} is the [SS10] master secret key for iteration $i+1$.

Our change will be to eliminate g in its entirety, by modifying U as below. Here, we expand out $\mathsf{msk}_{i+1} = \{\mathsf{sk}_{i+1,j,b}\}_{j,b}$ along with [SS10]'s key generation algorithm which simply outputs a selection of SKE keys. We use $(f, \widetilde{\mathsf{st}}_{i+1})[j]$ to denote the j^{th} bit of the tuple $(f, \widetilde{\mathsf{st}}_{i+1})$.

> $U[x_i, p_i, p_{i+1}, \{\mathsf{sk}_{i+1,j,b}\}_{j,b}](f, \widetilde{\mathsf{st}}_i)$:
>
> 1. $p_i = \widetilde{\mathsf{st}}_i \oplus p_i$.
> 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 $(y_i, \{\mathsf{sk}_{i+1,j,(f,\widetilde{\mathsf{st}}_{i+1})[j]}\}_{j,b})$.

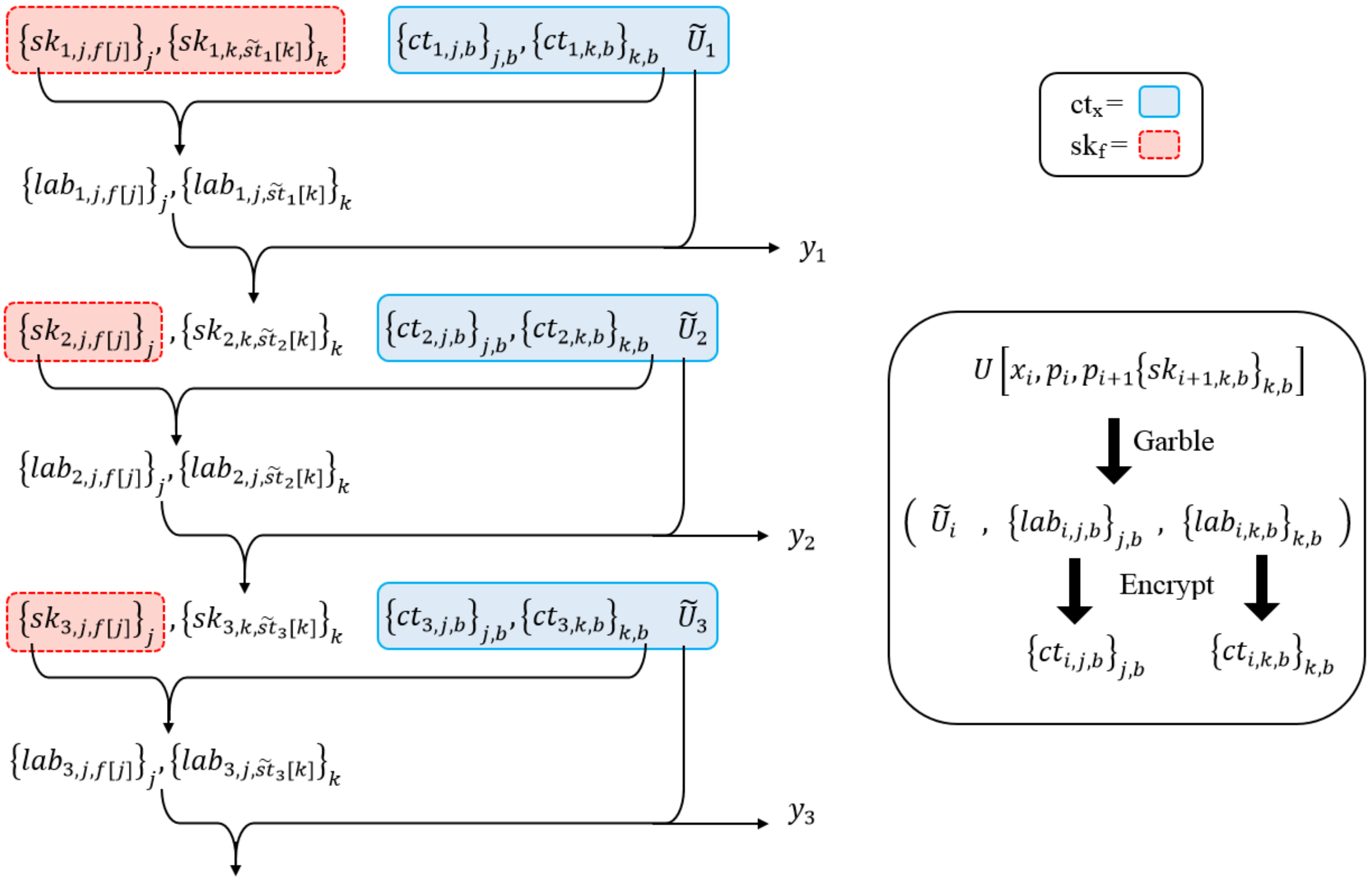
Observe that we no longer have any recursive function definitions! This is because our main function U is generated by the *encryptor*, and thus does not need to recursively generate copies of itself. The only thing that needs to be passed onto the next iteration are the SKE keys $\mathsf{sk}_{i+1,j,(f,\widetilde{\mathsf{st}}_i)[j]}$ representing the $(i+1)^{th}$ [SS10] function key, which can be of fixed size only dependent on the size of (f, st_i). Thus, we have fixed our parameter issues!

As one further optimization, we can modify U so that rather than outputting the SKE secret keys for both f and $\widetilde{\mathsf{st}}_{i+1}$, it only outputs the secret keys corresponding to $\widetilde{\mathsf{st}}_{i+1}$. Since the function f does not change between iterations, we can provide the SKE keys corresponding to f directly in the $\mathsf{One\text{-}sFE}$ function key for f. More precisely, the $\mathsf{One\text{-}sFE}$ function key for f will contain PRF keys which will allow the user to generate the corresponding SKE keys for f for every iteration i.

Final Construction. This gives us the following scheme, depicted in Figure 5. Let $\ell_{\mathcal{F}}$ and $\ell_{\mathcal{S}}$ be the lengths of the functions f and intermediate states st_i supported by our scheme, and let SKE be a secret-key encryption scheme.

- $\mathsf{Setup}(1^\lambda)$: Generate PRF keys for computing pads p_i along with $2(\ell_{\mathcal{F}} + \ell_{\mathcal{S}})$ SKE keys $\{\mathsf{sk}_{i,j,b}\}_{j\in[\ell_{\mathcal{F}}],b\in\{0,1\}}, \{\mathsf{sk}_{i,k,b}\}_{k\in[\ell_{\mathcal{S}}],b\in\{0,1\}}$ for every iteration i. Output these PRF keys as the master secret key MSK.

- $\mathsf{EncSetup}(\mathsf{MSK})$: Output $\mathsf{Enc.st} = \perp$.
- $\mathsf{Enc}(\mathsf{MSK}, \mathsf{Enc.st}, i, x_i)$:
  1. Use MSK to generate the $2(\ell_{\mathcal{F}} + \ell_{\mathcal{S}})$ SKE keys for indices i and $i+1$.
  2. Define circuit $U_i = U[x_i, p_i, p_{i+1}, \{\mathsf{sk}_{i+1,j,b}\}_{j,b}, \{\mathsf{sk}_{i+1,k,b}\}_{k,b}]$ as below where U_i has input x_i, the pads p_i and p_{i+1}, and the secret keys $\{\mathsf{sk}_{i+1,j,b}\}_{j\in[\ell_{\mathcal{F}}],b\in\{0,1\}}$, $\{\mathsf{sk}_{i+1,k,b}\}_{k\in[\ell_{\mathcal{S}}],b\in\{0,1\}}$ for the next iteration hardwired into it.
  3. Garble U_i using a circuit garbling scheme to get $\widetilde{U}_i$ and input labels $\{\mathsf{lab}_{i,j,b}\}_{j\in[\ell_{\mathcal{F}}],b\in\{0,1\}}$, $\{\mathsf{lab}_{i,k,b}\}_{k\in[\ell_{\mathcal{S}}],b\in\{0,1\}}$ where the first set of labels will correspond to the input wires for f and the second set of labels will correspond to the input wires for $\widetilde{\mathsf{st}}_i$.
  4. Encrypt each label $\mathsf{lab}_{i,j,b}$ under $\mathsf{sk}_{i,j,b}$ to get $\mathsf{ct}_{i,j,b}$. Similarly, encrypt each label $\mathsf{lab}_{i,k,b}$ under $\mathsf{sk}_{i,k,b}$ to get $\mathsf{ct}_{i,k,b}$.
  5. Output $\mathsf{CT}_i = (\widetilde{U}_i, \{\mathsf{ct}_{i,j,b}\}_{j,b}, \{\mathsf{ct}_{i,k,b}\}_{k,b})$.
- $\mathsf{KeyGen}(\mathsf{MSK}, f)$: Use MSK to compute the SKE keys $\{\mathsf{sk}_{i,k,\widetilde{\mathsf{st}}_1[k]}\}_{k\in[\ell_{\mathcal{S}}]}$ for the first encrypted state $\widetilde{\mathsf{st}}_1 = p_1 \oplus \mathsf{st}_1$. Then, use MSK to generate a limited selection of PRF keys which will allow the user to compute $\{\mathsf{sk}_{i,j,f[j]}\}_{j\in\ell_{\mathcal{F}}}$ for every iteration i, but no other SKE keys. Output these PRF keys along with the SKE keys for $\widetilde{\mathsf{st}}_1$ as the function key SK_f.
- $\mathsf{Dec}(\mathsf{SK}_f, \mathsf{Dec.st}_i, i, \mathsf{CT}_i)$: Use the SKE keys for f and $\widetilde{\mathsf{st}}_{i+1}$ provided in SK_f and/or $\mathsf{Dec.st}_i$ to decrypt the corresponding ciphertexts from CT_i and recover $\{\mathsf{lab}_{i,j,f[j]}\}_{j\in[\ell_{\mathcal{F}}]}, \{\mathsf{lab}_{i,k,\widetilde{\mathsf{st}}_i[k]}\}_{k\in[\ell_{\mathcal{S}}]}$. Then, evaluate the garbled circuit $\widetilde{U}_i$ from CT_i on these labels to learn $U_i(f, \widetilde{\mathsf{st}}_i) = (y_i, \{\mathsf{sk}_{i+1,k,\widetilde{\mathsf{st}}_{i+1}[k]}\}_k)$. Output y_i and $\mathsf{Dec.st}_{i+1} = \{\mathsf{sk}_{i+1,k,\widetilde{\mathsf{st}}_{i+1}[k]})\}_k$.



$U[x_i, p_i, p_{i+1}\{sk_{i+1,k,b}\}_{k,b}](f, \widetilde{\mathsf{st}}_i)$:

1. $p_i = \widetilde{\mathsf{st}}_i \oplus p_i$.
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 $(y_i, \{sk_{i+1,k,\widetilde{\mathsf{st}}_{i+1}[k]}\}_{k,b})$.

Figure 5: Construction of One-sFE.

Correctness follows from the correctness of the garbling scheme and the SKE scheme. For security, we will sequentially simulate each of the garbled circuits U_i starting from iteration 1. Observe that simulating the $(i-1)^{th}$ circuit will hide all of the values hardwired into U_{i-1}, including x_{i-1} and the SKE keys for the next iteration. Thus, after simulating iteration $i-1$, we can switch all ciphertexts $\{\mathsf{ct}_{i,j,1-f[j]}\}_j, \{\mathsf{ct}_{i,k,1-\widetilde{\mathsf{st}}[k]}\}_k$ for the unused labels $\{\mathsf{ct}_{i,j,1-f[j]}\}_j, \{\mathsf{ct}_{i,k,1-\widetilde{\mathsf{st}}[k]}\}_k$ to encryptions of $\perp$ since the corresponding secret keys are now kept hidden from the adversary. Then, we can use the security of the garbling scheme to simulate both the garbled circuit $\widetilde{U}_i$ for x_i and the input labels $\{\mathsf{lab}_{i,j,f[j]}\}_{j\in[\ell_{\mathcal{F}}]}, \{\mathsf{lab}_{i,j,\widetilde{\mathsf{st}}[j]}\}_{j\in[\ell_{\mathcal{S}}]}$ from $((f, \widetilde{\mathsf{st}}_i), U_i(f, \widetilde{\mathsf{st}}_i))$. Once we have simulated all of the circuits, then the entire stream $x = x_1 \ldots x_n$ will be hidden beyond what is learned from $f(x)$. Furthermore, even though we reveal the padded states $\widetilde{\mathsf{st}}_i$, the intermediate states st_i are also hidden, since the pads p_i embedded in each U_i are also hidden. Thus, we have the desired security.

Function-Selective Security. Our final One-sFE scheme inherits the function-selective security of [SS10], meaning that the scheme is only secure if the adversary makes its function query before its message queries. This is due to the selective security of our garbled circuits. It might be tempting to try to get adaptive security by either using an adaptive garbling scheme or to use a (single-key, single-ciphertext) adaptively secure FE scheme from OWFs, such as [GVW12], instead of [SS10]. However, there are some barriers to this approach. In particular, adaptive simulation-secure sFE is impossible even in the secret key setting and even when the adversary receives just one function key and one stream of challenge ciphertexts.[11] This means that any adaptive version of One-sFE would need to rely on additional techniques beyond just simulation techniques. However, we observe that both adaptive garbling schemes and (single-key, single-ciphertext) adaptively secure FE schemes like [GVW12] are *simulation*-secure, and thus we should expect to find additional problems if we directly try to insert them into our construction. Indeed, using these primitives will lead to parameter issues as adaptive garbling schemes have large input encodings, and [GVW12] has large function keys. Consequently, we leave adaptively secure One-sFE from OWFs (or any assumption weaker than $i\mathcal{O}$) as an interesting open problem.

2.2 Bootstrapping to a Q-Bounded Public-Key Streaming FE

We now adapt techniques from [AS16,GKS23] to bootstrap One-sFE to a Q-bounded, semi-adaptive-function-selective secure, *public-key* FE scheme.

Prior Work. Let us first review the prior work. [GKS23] show how to bootstrap a single-key, single-ciphertext, secret-key sFE scheme One-sFE into an *unbounded-collusion* public-key sFE scheme using *unbounded-collusion* public-key FE. At a high level, the idea is to generate a new One-sFE master secret key for every (function, stream) pair. This way, each One-sFE master secret key will only ever be used once, allowing us to use a reduction to the security of One-sFE.

In more detail, we will utilize three functional encryption schemes:

- One-sFE: the single-key, single-ciphertext, secret-key sFE scheme we wish to bootstrap.
- FE: an (unbounded-collusion) public-key FE scheme. This scheme will be responsible for generating a new One-sFE master secret key One-sFE.msk and a corresponding One-sFE function key $\mathsf{One\text{-}sFE.sk}_f$ for every (function, stream) pair.
- FPFE: an (unbounded-collusion) function-private, secret-key FE scheme.[12] This scheme will use the One-sFE master secret key generated by FE to encrypt each stream value x_i under One-sFE.

To perform the bootstrapping, we will do the following:[13]

[11] An adaptive simulation-secure scheme necessitates simulation of an unbounded number of ciphertexts (one for each element of the stream) without knowledge of any of the output values; followed by the simulation of an a priori bounded-size function key that provides the correct output values for all of the ciphertexts (c.f. [BSW11]).

[12] A function-private FE scheme hides both the messages and the functions. In the secret-key setting, we can build a function-private FE scheme from any standard FE scheme using the function-privacy transformation of [BS18].

[13] For ease of explanation, we have omitted some details from this construction. In particular, in the actual construction, G and H have additional branches of computation which are only ever used in the security proof.

- Our setup algorithm will generate FE keys $(\mathsf{FE.mpk}, \mathsf{FE.msk})$ which will be the master public key and master secret key of our scheme.

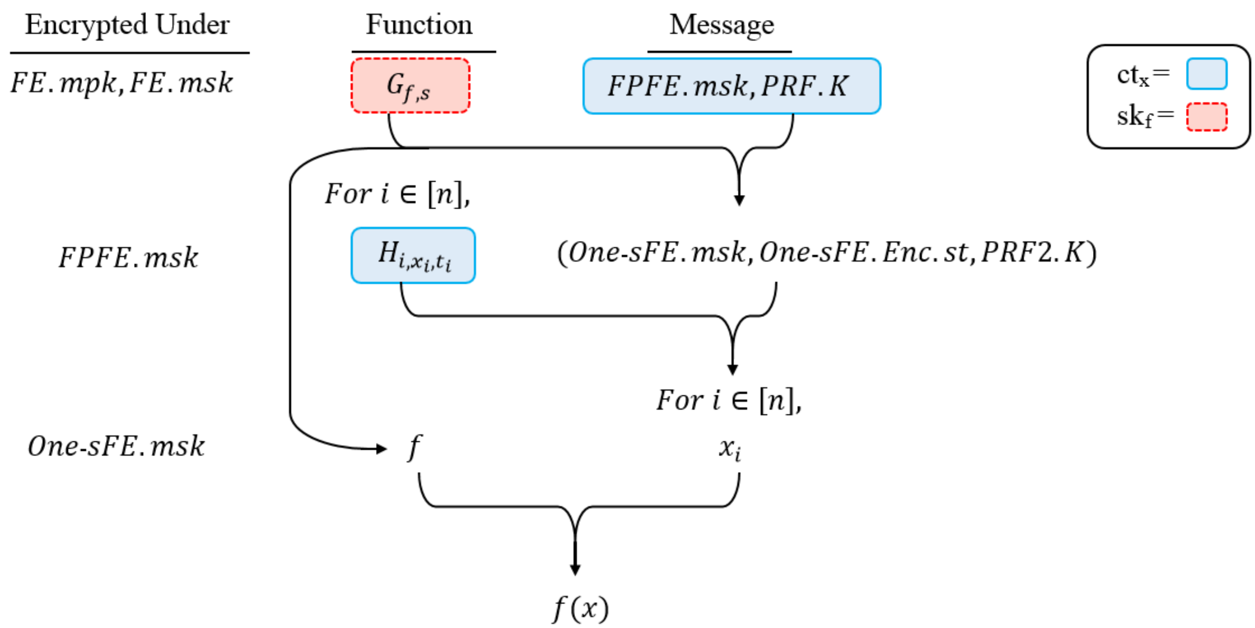
- To generate a function key for a streaming function f, we will create an FE function key $\mathsf{FE.sk}_G$ for the function $G_{f,s}$ defined below where s is a random value. $G_{f,s}$ will generate a fresh $\mathsf{One\text{-}sFE}$ master secret key $\mathsf{One\text{-}sFE.msk}$ along with a corresponding encryption state $\mathsf{One\text{-}sFE.Enc.st}$ and a corresponding function key $\mathsf{One\text{-}sFE.sk}_f$ for f. The output of $G_{f,s}$ will be $\mathsf{One\text{-}sFE.sk}_f$ and an FPFE ciphertext encrypting (1) $\mathsf{One\text{-}sFE.msk}$, (2) $\mathsf{One\text{-}sFE.Enc.st}$ and (3) a PRF key.

- To encrypt a stream $x = x_1 \dots x_n$:

  1. We will first create an FE ciphertext $\mathsf{FE.ct}$ of $(\mathsf{FPFE.msk}, \mathsf{PRF}.K)$ where $\mathsf{FPFE.msk}$ is an FPFE master secret key and $\mathsf{PRF}.K$ is a PRF key. We will provide $\mathsf{FE.ct}$ with the first ciphertext of our sFE scheme.
  2. Upon receiving the i^{th} stream value x_i, we will create and output an FPFE function key $\mathsf{FPFE.sk}_{H_i}$ for the function $H_i = H_{i,x_i,t_i}$ defined below where t_i is a random value. H_i will take as input a $\mathsf{One\text{-}sFE}$ master secret key (and a few other needed values) and output a $\mathsf{One\text{-}sFE}$ encryption $\mathsf{One\text{-}sFE.ct}_i$ of x_i.

- To decrypt:

  1. We can first use FE to decrypt $\mathsf{FE.ct}$ (from our first sFE ciphertext) with $\mathsf{FE.sk}_G$ (from our sFE function key) to get $\mathsf{One\text{-}sFE.sk}_f$ and an FPFE ciphertext $\mathsf{FPFE.ct}$.
  2. We can then use FPFE to decrypt $\mathsf{FPFE.ct}$ with $\mathsf{FPFE.sk}_{H_i}$ (from the i^{th} sFE ciphertext) to get $\mathsf{One\text{-}sFE.ct}_i$.
  3. Finally, we can use $\mathsf{One\text{-}sFE}$ to decrypt each $\mathsf{One\text{-}sFE.ct}_i$ using $\mathsf{One\text{-}sFE.sk}_f$ to learn the corresponding output values y_i.



$G_{f,s}(\mathsf{FPFE.msk}, \mathsf{PRF}.K)$:

1. $(r_{\mathsf{Setup}}, r_{\mathsf{EncSetup}}, r_{\mathsf{KeyGen}}, r_{\mathsf{PRF2}}, r_{\mathsf{Enc}}) \leftarrow \mathsf{PRF.Eval}(\mathsf{PRF}.K, s)$.
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 6: [GKS23]'s technique for bootstrapping to public-key sFE.

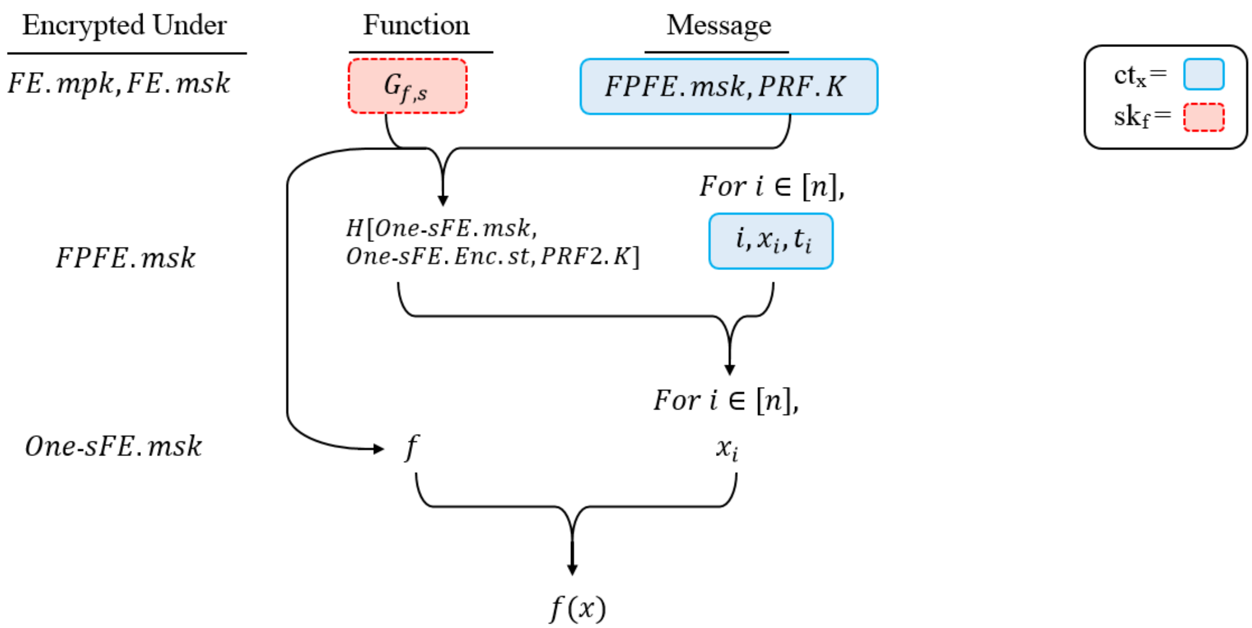
Correctness follows from the correctness of the underlying functional encryption schemes. For security, we will focus on each (function,stream) pair at a time to define the hybrids. We will first program the output values $(\mathsf{One\text{-}sFE.sk}_f, \mathsf{FPFE.ct})$ inside the function key for $G_{f,s}$ as part of an SKE ciphertext, where $\mathsf{FPFE.ct}$ is encrypting the tuple $(\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, \mathsf{PRF2}.K)$, and put the corresponding SKE secret key inside the FE ciphertext output during encryption of the first block. This allows us to get rid of the values $\mathsf{FPFE.msk}$ and $\mathsf{PRF}.K$ from the FE ciphertext, so that we can now use FPFE security to hardwire the $\mathsf{One\text{-}sFE}$ ciphertext, encrypting the i^{th} input block x_i, inside the FPFE function key for function H_i. This is to remove the values $(\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, \mathsf{PRF2}.K)$ from the FPFE ciphertext and x_i from the FPFE function key for H_i, so that we can now use $\mathsf{One\text{-}sFE}$ security and make the final switch for this (function,stream) pair.

Adapting to the bounded-collusion setting. As a first natural idea, we will try replacing each of the (unbounded-collusion) schemes in the bootstrapping with Q-bounded schemes which can be built from PKE using the work of [AV19].

Let us count the number of FE and FPFE function keys that will be generated during the security game of the Q-bounded sFE scheme. For our FE scheme, we see that we only need to generate one FE key per function key of our sFE scheme. Thus, a collusion-bound on the FE scheme would match the collusion-bound of our $\mathsf{One\text{-}sFE}$ scheme. Unfortunately, for our FPFE scheme, we need a number of function keys each to the length n of the challenge stream. This can be an arbitrary polynomial which may be larger than our collusion-bound Q.

Our key observation is that while we may need to generate many FPFE function keys, we actually only need to generate one FPFE *ciphertext* per (function, stream) pair (or equivalently, one FPFE ciphertext per function per $\mathsf{FPFE.msk}$). Then, since a function-private FE scheme is symmetric with respect to the hiding properties provided for the function and the message, we can solve our issues by swapping the roles of the ciphertexts and the function keys in our FPFE scheme. This gives us the scheme depicted in Figure 7.

This means that for each function key in our sFE scheme, we will only need one FE and FPFE function key per corresponding FE or FPFE master secret key. Thus, in the security proof of our Q-bounded sFE scheme, it suffices for the FE and FPFE schemes to be Q-bounded since we will never need more than Q function keys per schemes. As Q-bounded FE and FPFE can be built from PKE, then the bootstrapping step only requires PKE. This completes our construction.



$G_{f,s}(\mathsf{FPFE.msk}, \mathsf{PRF}.K)$:

1. $(r_{\mathsf{Setup}}, r_{\mathsf{EncSetup}}, r_{\mathsf{KeyGen}}, r_{\mathsf{PRF2}}, r_{\mathsf{Enc}}) \leftarrow \mathsf{PRF.Eval}(\mathsf{PRF}.K, s)$.
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. $\mathsf{FPFE.sk}_H \leftarrow \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H[\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, \mathsf{PRF2}.K]; r_{\mathsf{Enc}})$.
7. Output $(\mathsf{One\text{-}sFE.sk}_f, \mathsf{FPFE.sk}_H)$.

$H[\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, \mathsf{PRF2}.K](i, x_i, t_i)$:

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 7: Our technique for bootstrapping to public-key sFE.

3 Preliminaries

Throughout, we will use λ to denote the 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 use the standard definitions of one way functions (OWFs), pseudorandom functions (PRFs), and secret-key encryption (SKE) with pseudorandom ciphertexts. We formally define the latter two notions in Appendix A.1.

Definition 3.1 (Garbling Scheme)**.** *A garbling scheme is a tuple of* PPT *algorithms* $\mathsf{GC} = (\mathsf{Garble}, \mathsf{Eval})$ *defined as follows:*

- $\mathsf{Garble}(1^\lambda, C)$*: takes as input the security parameter λ and a circuit C with n-bit inputs, and outputs a garbled circuit $\widetilde{C}$ and input labels $\{\mathsf{lab}_{k,b}\}_{k\in[n],b\in\{0,1\}}$ where each label $\mathsf{lab}_{k,b} \in \{0,1\}^\lambda$.*
- $\mathsf{Eval}(\widetilde{C}, \{\mathsf{lab}_k\}_{k\in[n]})$*: takes as input a garbled circuit $\widetilde{C}$ and input labels $\{\mathsf{lab}_k\}_{k\in[n]}$, and outputs a value y.*

Correctness: *For all $\lambda \in \mathbb{N}$, all circuits C with n-bit inputs, and all inputs $x \in \{0,1\}^n$,*

$$\Pr[\mathsf{Eval}(\widetilde{C}, \{\mathsf{lab}_{k,x[k]}\}_{k\in[n]}) = C(x) : (\widetilde{C}, \{\mathsf{lab}_{k,b}\}_{k\in[n],b\in\{0,1\}}) \leftarrow \mathsf{Garble}(1^\lambda, C)] = 1$$

where $x[k]$ denotes the k^{th} bit of x.

Selective Simulation Security: *There exists a* PPT *simulator* Sim *and a negligible function μ such that for all* PPT *adversaries $\mathcal{A}$ and all $\lambda \in \mathbb{N}$,*

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

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

$\mathsf{Expt}^{\mathsf{GC\text{-}Sel\text{-}SIM}}_{\mathcal{A}}(1^\lambda, b)$*:*

1. *$\mathcal{A}$ takes as input 1^λ and outputs (C, x) where C is a circuit with n-bit inputs and $x \in \{0,1\}^n$.*
2. *If $i = 0$, $(\widetilde{C}, \{\mathsf{lab}_{k,b}\}_{k\in[n],b\in\{0,1\}}) \leftarrow \mathsf{Garble}(1^\lambda, C)$.*
3. *If $i = 1$, $(\widetilde{C}, \{\mathsf{lab}_{k,x[k]}\}_{k\in[n]}) \leftarrow \mathsf{Sim}(1^\lambda, 1^{|C|}, x, C(x))$.*
4. *Send $(\widetilde{C}, \{\mathsf{lab}_{k,x[k]}\}_{k\in[n]})$ to $\mathcal{A}$.*

5. $\mathcal{A}$ outputs b' which is the output of the experiment.

Lemma 3.2 ([Yao86]). *If there exist* OWF*s, then there exists a garbling scheme.*

3.1 Functional Encryption

Here we provide some fundamental definitions for functional encryption (FE) schemes. In this paper, we focus on Q-bounded FE schemes in which the adversary is restricted to obtaining at most Q functional keys. [AV19] show how to build such schemes from minimal assumptions.

Theorem 3.3 ([AV19]). *Assuming the existence of a public-key (resp. secret-key) encryption scheme, there exists a Q-bounded, adaptive-IND-secure, public-key (resp. secret-key)* FE *scheme for* P/Poly.

To define FE for P/Poly, we first define a class of functions parameterized by function size, input length, and output length.

Definition 3.4 (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 3.5 (Public-Key Functional Encryption). *A public-key functional encryption scheme for* P/Poly *is a tuple of PPT algorithms* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ *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 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 $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$,*

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

Definition 3.6 (Q-Bounded, Adaptive-IND-Security for Public-Key FE). *A public-key functional encryption scheme* FE *for* P/Poly *is Q-bounded, adaptive-IND-secure if there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$ and every PPT adversary $\mathcal{A}$,*

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

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

[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.

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

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***: $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}})$.
3. ***Public Key***: *Send* mpk *to* $\mathcal{A}$.
4. *For a polynomial number of rounds, the adversary can do either one of the following in each round:*
   (a) ***Function Query***: *The adversary can make at most $Q = Q(\lambda)$ such queries:*
      i. *$\mathcal{A}$ outputs a function query $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$*
      ii. $\mathsf{sk}_f \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}, f)$
      iii. *Send* sk_f *to* $\mathcal{A}$
   (b) ***Challenge Message Query***: *The adversary can make at most one such query.*
      i. *$\mathcal{A}$ outputs a challenge message pair (x_0, x_1) where $x_0, x_1 \in \{0,1\}^{\ell_{\mathcal{X}}}$.*
      ii. $\mathsf{ct} \leftarrow \mathsf{FE.Enc}(\mathsf{mpk}, x_b)$
      iii. *Send* ct *to* $\mathcal{A}$.
5. ***Experiment Outcome***: *$\mathcal{A}$ outputs a bit b' which is the output of the experiment.*

Additionally, when running the experiment, we immediately halt and output 0 if the adversary ever aborts or if it at any point $f(x_0) \neq f(x_1)$ for some message query (x_0, x_1) and function query f submitted by the adversary.

Definition 3.7 (Other Public-Key FE Security Definitions)**.** *There are many variations of the security definition. We list a few below:*

- ***Q-Bounded, Semi-Adaptive-IND-Security***: *The adversary is required to make the message query before the function queries. This is identical to Definition 3.6, except that we do not allow the adversary to make a* Challenge Message Query *after it has made a* Function Query.
- ***Q-Bounded, Semi-Adaptive-Function-Selective-IND-Security:*** *The adversary is required to make all function queries before the message query. This is identical to Definition 3.6, except that we do not allow the adversary to make a* Function Query *after it has made a* Challenge Message Query.
- ***Q-Bounded, Selective-IND-Security***: *The adversary is required to make the message query at the beginning of the experiment. This is similar to Definition 3.6, except that we allow the adversary to made a* Challenge Message Query *in between the* Setup *step and the* Public Key *step, but do not allow the adversary to make any* Challenge Message Queries *after the* Public Key *step.*
- ***Q-Bounded, Function-Selective-IND-Security:*** *The adversary is required to make all function queries at the beginning of the experiment. This is similar to Definition 3.6, except that we allow the adversary to make up to Q* Function Queries *in between the* Setup *step and the* Public Key *step, but do not allow the adversary to make any* Function Queries *after the* Public Key *step.*

3.1.2 Secret-Key Functional Encryption

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

Definition 3.8 (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.*

Remark 3.9 (Security Definitions). 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 make multiple challenge message queries. We formally define these security definitions in Appendix A.2.

Remark 3.10 (Function Privacy). In the secret-key setting, we can also achieve a notion of function privacy. We defer this definition to Appendix A.2.

3.2 Streaming Functional Encryption

Guan, Korb, and Sahai [GKS23] define streaming functional encryption (sFE) as functional cncryption (FE) for a class of *streaming functions*. In this paper, we focus on Q-bounded sFE schemes in which the adversary is restricted to obtaining at most Q functional keys.

3.2.1 Streaming Functions

Definition 3.11 (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 $x = x_1 \ldots x_n \in \mathcal{X}^n$ (denoted $f(x)$) to be $y = y_1 \ldots y_n \in \mathcal{Y}^n$ where*[15] *we have $\mathsf{st}_1 = \perp$ and*

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

Definition 3.12 (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 $\mathcal{Y} = \{0,1\}^{\ell_\mathcal{Y}}$.*

3.2.2 Public Key Streaming Function Encryption

Following the syntax of standard FE, we define public key sFE as follows.

Definition 3.13 (Public-Key Streaming FE). *A public-key streaming functional encryption scheme for* P/Poly *is a tuple of PPT algorithms* $\mathsf{sFE} = (\mathsf{Setup}, \mathsf{EncSetup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ *defined as follows:*[16]

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

[15] We assume that all streaming functions have the same starting state $\perp$ (or the all zero string) which is included in their state space. Note that we can still begin computing from any arbitrary starting value by simply hardwiring that value into the description of our streaming function.

[16] We also allow Enc, EncSetup, 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.

- $\mathsf{EncSetup}(\mathsf{mpk})$: *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 *must be* ***streaming efficient****, meaning 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* $\mathsf{poly}(\lambda, \ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}})$.

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[17]

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

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

Definition 3.14 (Q-Bounded, Adaptive-IND-Security for Public-Key sFE)**.** *A public-key streaming FE scheme* sFE *for* $\mathsf{P/Poly}$ *is* Q*-bounded, adaptive-IND-secure if there exists a negligible function* μ *such that for all* $\lambda \in \mathbb{N}$ *and all PPT adversaries* $\mathcal{A}$,

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

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

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

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. ***Setup****:* $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{sFE.Setup}(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}})$.
3. ***Public Key****: Send* mpk *to* $\mathcal{A}$.
4. *For a polynomial number of rounds, the adversary can do either one of the following in each round:*
   (a) ***Function Query****: The adversary can make at most* $Q = Q(\lambda)$ *such queries:*

[17] As with all streaming functions, we assume that $\mathsf{Dec.st}_1 = \perp$ if not otherwise specified.

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

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

iii. *Send* sk_f *to* $\mathcal{A}$.

(b) ***Challenge Message Query:***

i. *If this is the first challenge message query, sample* $\mathsf{Enc.st} \leftarrow \mathsf{sFE.EncSetup}(\mathsf{mpk})$ *and initialize the index* $i = 1$. *Else, increment the index* i *by* 1.

ii. $\mathcal{A}$ *outputs a challenge message pair* $(x_i^{(0)}, x_i^{(1)})$ *where* $x_i^{(0)}, x_i^{(1)} \in \{0,1\}^{\ell_{\mathcal{X}}}$.

iii. $\mathsf{ct}_i \leftarrow \mathsf{sFE.Enc}(\mathsf{mpk}, \mathsf{Enc.st}, i, x_i^{(b)})$.

iv. *Send* ct_i *to* $\mathcal{A}$.

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

Additionally, when running the experiment, we immediately halt and output 0 *if the adversary ever aborts or if it at any point some function query* f *submitted by the adversary yields different outputs on the challenge message streams submitted so far (i.e. if* $f(x^{(0)}) \neq f(x^{(1)})$ *for some function query* f *submitted by the adversary where* $\{(x_i^{(0)}, x_i^{(1)})\}_{i\in[t]}$ *are the message queries submitted so far,* $x^{(0)} = x_1^{(0)} \dots x_t^{(0)}$*, and* $x^{(1)} = x_1^{(1)} \dots x_t^{(1)}$*).*

Definition 3.15 (Other Public-Key sFE Security Definitions)**.** *There are many variations of the security definition. We list a few below:*

- Q***-Bounded, Semi-Adaptive-IND-Security****: The adversary is required to make all message queries before any function queries. This is identical to Definition 3.14, except that we do not allow the adversary to make a* Challenge Message Query *after it has made a* Function Query*.*

- Q***-Bounded, Semi-Adaptive-Function-Selective-IND-Security:*** *The adversary is required to make all function queries before any message queries. This is identical to Definition 3.14, except that we do not allow the adversary to make a* Function Query *after it has made a* Challenge Message Query*.*

- Q***-Bounded, Selective-IND-Security:*** *The adversary is required to make all message queries at the beginning of the experiment. This is similar to Definition 3.14, except that we allow the adversary to make a polynomial number of* Challenge Message Queries *in between the* Setup *step and the* Public Key *step, but do not allow the adversary to make any* Challenge Message Queries *after the* Public Key *step.*

- Q***-Bounded, Function-Selective-IND-Security:*** *The adversary is required to make all function queries at the beginning of the experiment. This is similar to Definition 3.14, except that we allow the adversary to make up to* Q Function Queries *in between the* Setup *step and the* Public Key *step, but do not allow the adversary to make any* Function Queries *after the* Public Key *step.*

3.2.3 Secret-Key Streaming Functional Encryption

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

Definition 3.16 (Secret-Key Streaming Functional Encryption)**.** *Secret-key* sFE *is the same as public-key* sFE *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.*

Remark 3.17 (Security Definitions)**.** 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 pairs of challenge streams. We formally define these security definitions in Appendix A.3.

Definition 3.18 (Single-Key, Single-Ciphertext Security)**.** *In our security definitions, we may use the modifier "single-key, single-ciphertext" instead of "Q-bounded". 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 (i.e.* 1*-bounded) and submitting only one pair of challenge message streams (though each stream may consist of many elements) in the relevant security game.*

Remark 3.19 (Simulation Security)**.** We will also define a weak notion of simulation security in the secret-key setting. We formally define this in Appendix A.3.

4 Single-Key, Single-Ciphertext, 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 4.1. *Assuming* OWF*s, there exists a single-key, single-ciphertext, function-selective-SIM-secure, secret-key sFE scheme for* P/Poly.

To prove Theorem 4.1, we build an sFE scheme from the following tools, which can each be built from OWFs using standard techniques. [Gol01, Gol09, Yao86]

Tools.

- $\mathsf{PRF}_1, \mathsf{PRF}_2, \mathsf{PRF}_3, \mathsf{PRF}_p$: Secure pseudorandom function families where $\mathsf{PRF}_c = (\mathsf{PRF}_c.\mathsf{Setup}, \mathsf{PRF}_c.\mathsf{Eval})$ for all $c \in \{1, 2, 3, p\}$.
- $\mathsf{SKE} = (\mathsf{SKE.Setup}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$: A secure secret-key encryption scheme.
- $\mathsf{GC} = (\mathsf{GC.Garble}, \mathsf{GC.Eval})$: A secure garbling scheme.

4.1 Parameters

On security parameter 1^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:

- We instantiate our PRFs with the following parameters:

	Security Parameter	Input Size	Output Size
PRF_1	λ	$\log(\ell_{\mathcal{F}}) + 1$	λ
PRF_2	λ	λ	λ
PRF_3	λ	$\lambda + \log(\ell_{\mathcal{S}}) + 1$	λ
PRF_p	λ	λ	$\ell_{\mathcal{S}}$

- SKE: We instantiate SKE with security parameter λ. We will use SKE to encrypt messages of length λ.
- GC: We instantiate GC with security parameter λ. We will use GC to garble circuits of the form $U[x_i, p_i, p_{i+1}, \{\mathsf{sk}'_{i+1,k,b}\}_{k\in[\ell_{\mathcal{S}}], b\in\{0,1\}}]$ as defined in Figure 8 where $x_i \in \{0,1\}^{\ell_{\mathcal{X}}}$, $p_i, p_{i+1} \in \{0,1\}^{\ell_{\mathcal{S}}}$, each $\mathsf{sk}'_{i+1,k,b}$ is a secret key of SKE, and which takes inputs of size $\ell_{\mathcal{F}} + \ell_{\mathcal{S}}$. Let ℓ_U be the size of such circuits. Recall that each of the input labels of a garbled circuit are size λ.

Remark 4.2. 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 can be implemented with OWFs and 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).

4.2 Construction

We now construct our single-key, single-ciphertext streaming functional encryption scheme $\mathsf{One\text{-}sFE}$.

Notation For notational convenience, when the parameters are understood, we will often omit the security, input size, and output size parameters from our algorithms.

- $\mathsf{One\text{-}sFE.Setup}(1^\lambda, 1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{S}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}})$:
  1. $K \leftarrow \mathsf{PRF}_1.\mathsf{Setup}(1^\lambda)$, $K' \leftarrow \mathsf{PRF}_3.\mathsf{Setup}(1^\lambda)$, $K_p \leftarrow \mathsf{PRF}_p.\mathsf{Setup}(1^\lambda)$.

  * Throughout, for $i \in [2^\lambda]$, $j \in [\ell_\mathcal{F}]$, $k \in [\ell_\mathcal{S}]$, $b \in \{0,1\}$, we will define

  $$\begin{aligned} K_{j,b} &= \mathsf{PRF}_2.\mathsf{Setup}(1^\lambda; \mathsf{PRF}_1.\mathsf{Eval}(K, (j,b))) \\ \mathsf{sk}_{i,j,b} &= \mathsf{SKE.Setup}(1^\lambda; \mathsf{PRF}_2.\mathsf{Eval}(K_{j,b}, i)) \\ \mathsf{sk}'_{i,k,b} &= \mathsf{SKE.Setup}(1^\lambda; \mathsf{PRF}_3.\mathsf{Eval}(K', (i,k,b))) \\ p_i &= \mathsf{PRF}_p.\mathsf{Eval}(K_p, i) \end{aligned}$$

  Observe that these values can be computed from i, j, k, b, K, K', and K_p.
  2. Output $\mathsf{MSK} = (K, K', K_p)$.

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

- $\mathsf{One\text{-}sFE.Enc}(\mathsf{MSK}, \mathsf{Enc.st}, i, x_i)$:
  1. Parse $\mathsf{MSK} = (K, K', K_p)$.
  2. **Garble circuit:**
     (a) Compute $p_i, p_{i+1}, \{\mathsf{sk}'_{i+1,k,b}\}_{k\in[\ell_\mathcal{S}], b\in\{0,1\}}$ from K', K_p.
     (b) Let $U_i = U[x_i, p_i, p_{i+1}, \{\mathsf{sk}'_{i+1,k,b}\}_{k\in[\ell_\mathcal{S}], b\in\{0,1\}}]$ as defined in Figure 8.
     (c) $(\widetilde{U}_i, \{\mathsf{lab}_{i,j,b}\}_{j\in[\ell_\mathcal{F}], b\in\{0,1\}}, \{\mathsf{lab}'_{i,k,b}\}_{k\in[\ell_\mathcal{S}], b\in\{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, U_i)$.[18]
  3. **Encrypt function labels:** For $j \in [\ell_\mathcal{F}]$, $b \in \{0,1\}$,
     (a) Compute $\mathsf{sk}_{i,j,b}$ from K.
     (b) $\mathsf{ct}_{i,j,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{i,j,b}, \mathsf{lab}_{i,j,b})$.
  4. **Encrypt state labels:** For $k \in [\ell_\mathcal{S}]$, $b \in \{0,1\}$,
     (a) Compute $\mathsf{sk}'_{i,k,b}$ from K'.
     (b) $\mathsf{ct}'_{i,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}'_{i,k,b}, \mathsf{lab}'_{i,k,b})$.
  5. Output $\mathsf{CT}_i = (\widetilde{U}_i, \{\mathsf{ct}_{i,j,b}\}_{j\in[\ell_\mathcal{F}], b\in\{0,1\}}, \{\mathsf{ct}'_{i,k,b}\}_{k\in[\ell_\mathcal{S}], b\in\{0,1\}})$.

[18] For notational convenience, we split the input labels for $\widetilde{U}_i$ into two categories depending upon what part of the input they represent. We use $\{\mathsf{lab}_{i,j,b}\}_{j\in[\ell_\mathcal{F}], b\in\{0,1\}}$ to refer to the labels for the part of the input representing f (i.e. the first $\ell_\mathcal{F}$ bits of the input), and use $\{\mathsf{lab}'_{i,k,b}\}_{k\in[\ell_\mathcal{S}], b\in\{0,1\}}$ to refer to the labels for the part of the input representing $\widetilde{\mathsf{st}}_i$ (i.e. the last $\ell_\mathcal{S}$ bits of the input).

$U[x_i, p_i, p_{i+1}, \{\mathsf{sk}'_{i+1,k,b}\}_{k\in[\ell_\mathcal{S}],b\in\{0,1\}}](f, \widetilde{\mathsf{st}}_i)$:

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 $(y_i, (\widetilde{\mathsf{st}}_{i+1}, \{\mathsf{sk}'_{i+1,k,\widetilde{\mathsf{st}}_{i+1}[k]}\}_{k\in[\ell_\mathcal{S}]}))$.

Figure 8: Definition of $U[x_i, p_i, p_{i+1}, \{\mathsf{sk}'_{i+1,k,b}\}_{k\in[\ell_\mathcal{S}],b\in\{0,1\}}]$

- $\mathsf{One\text{-}sFE.KeyGen}(\mathsf{MSK}, f)$
  1. Parse $\mathsf{MSK} = (K, K', K_p)$.
  2. **Compute PRF keys for generating SKE keys for f:**
     (a) Compute $\{K_{j,f[j]}\}_{j\in[\ell_\mathcal{F}]}$ from K.
  3. **Compute SKE keys for $\widetilde{\mathsf{st}}_1$:**
     (a) Compute p_1 from K_p.
     (b) $\widetilde{\mathsf{st}}_1 = p_1$.
     (Here, we assume $\mathsf{st}_1 = 0^{\ell_\mathcal{S}}$ for all streaming functions so that $\widetilde{\mathsf{st}}_1 = \mathsf{st}_1 \oplus p_1 = p_1$.)
     (c) Compute $\{\mathsf{sk}'_{1,k,\widetilde{\mathsf{st}}_1[k]}\}_{k\in[\ell_\mathcal{S}]}$ from K'.
  4. Output $\mathsf{SK}_f = (f, \{K_{j,f[j]}\}_{j\in[\ell_\mathcal{F}]}, (\widetilde{\mathsf{st}}_1, \{\mathsf{sk}'_{1,k,\widetilde{\mathsf{st}}_1[k]}\}_{k\in[\ell_\mathcal{S}]}))$.

- $\mathsf{One\text{-}sFE.Dec}(\mathsf{SK}_f, \mathsf{Dec.st}_i, i, \mathsf{CT}_i)$:
  1. Parse $\mathsf{SK}_f = (f, \{K_{j,f[j]}\}_{j\in[\ell_\mathcal{F}]}, (\widetilde{\mathsf{st}}_1, \{\mathsf{sk}'_{1,k,\widetilde{\mathsf{st}}_1[k]}\}_{k\in[\ell_\mathcal{S}]}))$.
  2. If $i > 1$, parse $\mathsf{Dec.st}_i = (\widetilde{\mathsf{st}}_i, \{\mathsf{sk}'_{i,k,\widetilde{\mathsf{st}}_i[k]}\}_{k\in[\ell_\mathcal{S}]})$.
  3. Parse $\mathsf{CT}_i = (\widetilde{U}_i, \{\mathsf{ct}_{i,j,b}\}_{j\in[\ell_\mathcal{F}],b\in\{0,1\}}, \{\mathsf{ct}'_{i,k,b}\}_{k\in[\ell_\mathcal{S}],b\in\{0,1\}})$.
  4. **Recover function labels:** For $j \in [\ell_\mathcal{F}]$,
     (a) $r_{i,j,f[j]} = \mathsf{PRF}_2.\mathsf{Eval}(K_{j,f[j]}, i)$.
     (b) $\mathsf{sk}_{i,j,f[j]} = \mathsf{SKE.Setup}(1^\lambda; r_{i,j,f[j]})$.
     (c) $\mathsf{lab}_{i,j,f[j]} = \mathsf{SKE.Dec}(\mathsf{sk}_{i,j,f[j]}, \mathsf{ct}_{i,j,f[j]})$.
  5. **Recover state labels:** For $k \in [\ell_\mathcal{S}]$,
     (a) $\mathsf{lab}'_{i,k,\widetilde{\mathsf{st}}_i[k]} = \mathsf{SKE.Dec}(\mathsf{sk}'_{i,k,\widetilde{\mathsf{st}}_i[k]}, \mathsf{ct}'_{i,k,\widetilde{\mathsf{st}}_i[k]})$.
  6. **Evaluate garbled ciruit:**
     (a) $(y_i, (\widetilde{\mathsf{st}}_{i+1}, \{\mathsf{sk}'_{i+1,k,\widetilde{\mathsf{st}}_{i+1}[k]}\}_{k\in[\ell_\mathcal{S}]})) = \mathsf{GC.Eval}(\widetilde{U}_i, \{\mathsf{lab}_{i,j,f[j]}\}_{j\in[\ell_\mathcal{F}]}, \{\mathsf{lab}'_{i,k,\widetilde{\mathsf{st}}_i[k]}\}_{k\in[\ell_\mathcal{S}]})$.
  7. $\mathsf{Dec.st}_{i+1} = (\widetilde{\mathsf{st}}_{i+1}, \{\mathsf{sk}'_{i+1,k,\widetilde{\mathsf{st}}_{i+1}[k]}\}_{k\in[\ell_\mathcal{S}]})$.
  8. Output $(y_i, \mathsf{Dec.st}_{i+1})$.

4.3 Correctness and Efficiency

Efficiency: Using our discussion above on parameters, it is easy to see that the size and runtime of all algorithms of our $\mathsf{One\text{-}sFE}$ scheme on security parameter 1^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: The i^{th} ciphertext consists of a garbled circuit for U_i (which has x_i embedded within it) along with encryptions of each of the input labels for the garbled circuit. Using the function key for f, we can recover the input labels corresponding to input f. Using the decryption state (or the function key if $i = 1$), we can recover the input labels corresponding to an encryption $\widetilde{\mathsf{st}}_i$ of st_i. Then, we can use the garbled circuit to evaluate $U_i(f, \widetilde{\mathsf{st}}_i)$ which gives us y_i, an encryption $\widetilde{\mathsf{st}}_{i+1}$ of st_{i+1} where $f(x_i, \mathsf{st}_i) = (y_i, \mathsf{st}_{i+1})$, and keys for later recovering the input labels corresponding to $\widetilde{\mathsf{st}}_{i+1}$. This gives us the desired output.

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 stream x where $x = x_1 \dots x_n$ for some $n \in [2^\lambda]$ and where each $x_i \in \{0,1\}^{\ell_{\mathcal{X}}}$.

The function key SK_f consists of

- The function f
- PRF_2 keys $\{K_{j,f[j]}\}_{j \in [\ell_{\mathcal{F}}]}$ for computing SKE keys $\{\mathsf{sk}_{i,j,f[j]}\}_{j \in [\ell_{\mathcal{F}}]}$ corresponding to f.
- A (one-time-pad) encryption $\widetilde{\mathsf{st}}_1$ of state st_1
- SKE keys $\{\mathsf{sk}_{1,k,\widetilde{\mathsf{st}}_1[k]}\}_{k \in [\ell_{\mathcal{S}}]}$ corresponding to $\widetilde{\mathsf{st}}_1$.

The i^{th} ciphertext CT_i consists of

- A garbled circuit $\widetilde{U}_i$ for the circuit $U[x_i, p_i, p_{i+1}, \{\mathsf{sk}_{i+1,k,b}\}_{k \in [\ell_{\mathcal{S}}], b \in \{0,1\}}]$ depicted in Figure 8.
- Ciphertexts $\mathsf{ct}_{i,j,b}$ encrypting labels $\mathsf{lab}_{i,j,b}$ for each $j \in [\ell_{\mathcal{F}}], b \in \{0,1\}$.
- Ciphertexts $\mathsf{ct}'_{i,k,b}$ encrypting labels $\mathsf{lab}'_{i,k,b}$ for each $k \in [\ell_{\mathcal{S}}], b \in \{0,1\}$.

Thus, we can prove by induction on i starting with $i = 1$ that

$$
\begin{aligned}
&\mathsf{One\text{-}sFE.Dec}(\mathsf{SK}_f, \mathsf{Dec.st}_i, i, \mathsf{CT}_i) \\
&= \mathsf{GC.Eval}(\widetilde{U}_i, \{\mathsf{SKE.Dec}(\mathsf{sk}_{i,j,f[j]}, \mathsf{ct}_{i,j,f[j]})\}_{j \in [\ell_{\mathcal{F}}]}, \{\mathsf{SKE.Dec}(\mathsf{sk}_{i,k,\widetilde{\mathsf{st}}_i[k]}, \mathsf{lab}'_{i,k,\widetilde{\mathsf{st}}_i[k]}\}_{k \in [\ell_{\mathcal{S}}]}) \\
&= \mathsf{GC.Eval}(\widetilde{U}_i, \{\mathsf{lab}_{i,j,f[j]}\}_{j \in [\ell_{\mathcal{F}}]}, \{\mathsf{lab}'_{i,k,\widetilde{\mathsf{st}}_i[k]}\}_{k \in [\ell_{\mathcal{S}}]}) \\
&= U[x_i, p_i, p_{i+1}, \{\mathsf{sk}_{i+1,k,b}\}_{k \in [\ell_{\mathcal{S}}], b \in \{0,1\}}](f, \widetilde{\mathsf{st}}_i) \\
&= (y_i, \mathsf{Dec.st}_i = (\widetilde{\mathsf{st}}_{i+1}, \{\mathsf{sk}'_{i+1,k,\widetilde{\mathsf{st}}_{i+1}[k]}\}_{k \in [\ell_{\mathcal{S}}]}))
\end{aligned}
$$

where $(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$, $\widetilde{\mathsf{st}}_{i+1}$ is a (one-time-pad) encryption of st_{i+1}, and $\{\mathsf{sk}_{i+1,k,\widetilde{\mathsf{st}}_{i+1}[k]}\}_{k \in [\ell_{\mathcal{S}}]}$ are SKE keys corresponding to $\widetilde{\mathsf{st}}_{i+1}$. Here, the first equality follows from the definition of $\mathsf{One\text{-}sFE.Dec}$ and our use of PRF_2 keys, the second equality follows by the correctness of SKE, the third equality follows by the correctness of GC, and the fourth equality follows from the definition of U. Thus, we get the correct output value y_i at each step.

4.4 Security

In this section, we prove that $\mathsf{One\text{-}sFE}$ is single-key, single-ciphertext, function-selective-SIM-secure.

4.4.1 Proof Overview

To build intuition, we provide a brief overview of each hybrid in our proof.

- $\mathbf{Hybrid}_0^{\mathcal{A}}(\lambda)$: This is the real world experiment where we use the algorithms defined in our construction.
- $\mathbf{Hybrid}_1^{\mathcal{A}}(\lambda)$: Here, we reorder several steps from the previous hybrid. In particular, the challenger now computes the random (one-time) pads p_i, the PRF keys, and the SKE keys earlier in the hybrid. For each message query x_i, the challenger also computes and stores the output value y_i and the updated state st_{i+1}. This hybrid is identically distributed to the previous hybrid.
- $\mathbf{Hybrid}_2^{\mathcal{A}}(\lambda)$: The outputs generated by the PRF keys K, K' and K_p are replaced with truly random strings. Indistinguishability follows from the security of $\mathsf{PRF}_1, \mathsf{PRF}_3$, and PRF_p.
- $\mathbf{Hybrid}_3^{\mathcal{A}}(\lambda)$: For each index i, we change how $\widetilde{\mathsf{st}}_i$ and p_i are generated. In the previous hybrids, we sampled a random p_i and set $\widetilde{\mathsf{st}}_i = \mathsf{st}_i \oplus p_i$. Now, we sample a random $\widetilde{\mathsf{st}}_i$ and set $p_i = \mathsf{st}_i \oplus \widetilde{\mathsf{st}}_i$. Since we have just swapped the roles of two random variables in an XOR equation, the two hybrids are identically distributed
- $\mathbf{Hybrid}_4^{\mathcal{A}}(\lambda)$: We remove usage of the PRF_2 keys corresponding to the negation of the bit-string for function f. More precisely, when answering the function query for f, the challenger only samples keys $\{K_{j,f[j]}\}_{j \in [\ell_{\mathcal{F}}]}$ and does not sample keys $\{K_{j,1-f[j]}\}_{j \in [\ell_{\mathcal{F}}]}$. When answering the challenge message queries, the challenger samples SKE keys $\mathsf{sk}_{i,j,1-f[j]}$, for $j \in [\ell_{\mathcal{F}}]$, by using true randomness instead of using $K_{j,1-f[j]}$ as was done previously. Indistinguishability follows by the security of PRF_2.
- $\mathbf{Hybrid}_5^{\mathcal{A}}(\lambda)$: We replace the ciphertexts $\mathsf{ct}_{i,j,1-f[j]}$ with encryptions of $\perp$. This removes the input labels corresponding to the negation of the bit-string for f from the adversary's view. This is feasible since the corresponding secret keys $\mathsf{sk}_{i,j,1-f[j]}$ are completely hidden from the adversary due to the change made in the previous hybrid. Indistinguishability follows from the security of SKE.
- We now go through the following hybrids for $\alpha \in [\mathsf{Bound}_{\mathcal{A}}]$ where $\mathsf{Bound}_{\mathcal{A}}$ is a bound on the runtime of $\mathcal{A}$, and thus an implicit bound on the number of challenge message queries made by the adversary. On iteration α, the goal is to switch to a hybrid where we simulate the α^{th} garbled circuit.
  - $\mathbf{Hybrid}_{6,\alpha,0}^{\mathcal{A}}(\lambda)$: For $i < \alpha$, we generate the garbled circuit $\widetilde{U}_i$ and the input labels corresponding to $(f, \widetilde{\mathsf{st}}_i)$ using the simulator for the garbling scheme. Since we are simulating the garbled circuit for $U_{\alpha-1}$, we no longer need to embed the secret keys $\mathsf{sk}'_{\alpha,k,1-\widetilde{\mathsf{st}}[\alpha]}$ into $U_{\alpha-1}$. Thus, we can replace the ciphertexts $\mathsf{ct}'_{\alpha,k,1-\widetilde{\mathsf{st}}[\alpha]}$ with encryptions of $\perp$. This removes the input labels corresponding to the negation of the bit-string for $\widetilde{\mathsf{st}}_\alpha$ from the adversary's view. For $\alpha = 1$, this hybrid is indistinguishable from $\mathbf{Hybrid}_5^{\mathcal{A}}$ by the security of SKE.

- $\mathbf{Hybrid}^{\mathcal{A}}_{6,\alpha,1}(\lambda)$: We generate the garbled circuit $\widetilde{U}_\alpha$ and the input labels corresponding to $(f, \widetilde{\mathsf{st}}_\alpha)$ using the simulator for the garbling scheme. This is feasible because we have already removed the α^{th} input labels for both the negation of the bit-string for f and the negation of the bit-string for $\widetilde{\mathsf{st}}_i$ from the adversary's view. Indistinguishability follows from the security of the garbling scheme. Additionally, the indistinguishability of $\mathbf{Hybrid}^{\mathcal{A}}_{6,\alpha,1}$ and $\mathbf{Hybrid}^{\mathcal{A}}_{6,\alpha+1,0}$ follows from the security of SKE.

- $\mathbf{Hybrid}^{\mathcal{A}}_{7}(\lambda)$: This is the ideal world experiment written using an explicit simulator Sim. This hybrid is identically distributed to $\mathbf{Hybrid}^{\mathcal{A}}_{6,\mathsf{Bound}_\mathcal{A},1}$ since $\mathbf{Hybrid}^{\mathcal{A}}_{6,\mathsf{Bound}_\mathcal{A},1}$ simulates every circuit U_i using the garbling scheme, and thus does not need to know any of the stream values x_i which were previously embedded in the circuits U_i.

4.4.2 Formal Proof

We now formally prove Theorem 4.1 via a hybrid argument.

$\mathbf{Hybrid}_0^{\mathcal{A}}(1^\lambda)$: This is the real world 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:** $K \leftarrow \mathsf{PRF}_1.\mathsf{Setup}(1^\lambda)$, $K' \leftarrow \mathsf{PRF}_3.\mathsf{Setup}(1^\lambda)$, $K_p \leftarrow \mathsf{PRF}_p.\mathsf{Setup}(1^\lambda)$.
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) **Compute PRF keys for generating SKE keys for f:**
      i. Compute $\{K_{j,f[j]}\}_{j\in[\ell_{\mathcal{F}}]}$ from K.
   (c) **Compute SKE keys for $\widetilde{\mathsf{st}}_1$:**
      i. Compute p_1 from K_p.
      ii. $\widetilde{\mathsf{st}}_1 = p_1$.
      iii. Compute $\{\mathsf{sk}'_{1,k,\widetilde{\mathsf{st}}_1[k]}\}_{k\in[\ell_{\mathcal{S}}]}$ from K'.
   (d) Send $\mathsf{SK}_f = (f, \{K_{j,f[j]}\}_{j\in[\ell_{\mathcal{F}}]}, (\widetilde{\mathsf{st}}_1, \{\mathsf{sk}'_{1,k,\widetilde{\mathsf{st}}_1[k]}\}_{k\in[\ell_{\mathcal{S}}]}))$ to $\mathcal{A}$.
4. **Challenge Message Queries:** For $i = 1, 2, 3, \ldots$
   (a) $\mathcal{A}$ outputs a challenge message $x_i \in \{0,1\}^{\ell_{\mathcal{X}}}$.
   (b) **Garble circuit:**
      i. Compute $p_i, p_{i+1}, \{\mathsf{sk}'_{i+1,k,b}\}_{k\in[\ell_{\mathcal{S}}],b\in\{0,1\}}$ from K', K_p.
      ii. Let $U_i = U[x_i, p_i, p_{i+1}, \{\mathsf{sk}'_{i+1,k,b}\}_{k\in[\ell_{\mathcal{S}}],b\in\{0,1\}}]$ as defined in Figure 8.
      iii. $(\widetilde{U}_i, \{\mathsf{lab}_{i,j,b}\}_{j\in[\ell_{\mathcal{F}}],b\in\{0,1\}}, \{\mathsf{lab}'_{i,k,b}\}_{k\in[\ell_{\mathcal{S}}],b\in\{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, U_i)$.
   (c) **Encrypt function labels:** For $j \in [\ell_{\mathcal{F}}]$, $b \in \{0,1\}$,
      i. Compute $\mathsf{sk}_{i,j,b}$ from K.
      ii. $\mathsf{ct}_{i,j,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{i,j,b}, \mathsf{lab}_{i,j,b})$.
   (d) **Encrypt state labels:** For $k \in [\ell_{\mathcal{S}}]$, $b \in \{0,1\}$,
      i. Compute $\mathsf{sk}'_{i,k,b}$ from K'.
      ii. $\mathsf{ct}'_{i,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}'_{i,k,b}, \mathsf{lab}'_{i,k,b})$.
   (e) Send $\mathsf{CT}_i = (\widetilde{U}_i, \{\mathsf{ct}_{i,j,b}\}_{j\in[\ell_{\mathcal{F}}],b\in\{0,1\}}, \{\mathsf{ct}'_{i,k,b}\}_{k\in[\ell_{\mathcal{S}}],b\in\{0,1\}})$ to $\mathcal{A}$.
5. **Experiment Outcome:** $\mathcal{A}$ outputs a bit b' which is the output of the experiment.

Hybrid$_1^{\mathcal{A}}(1^\lambda)$: We reorder several steps of the hybrid.

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:** $K \leftarrow \mathsf{PRF}_1.\mathsf{Setup}(1^\lambda)$, $K' \leftarrow \mathsf{PRF}_3.\mathsf{Setup}(1^\lambda)$, $K_p \leftarrow \mathsf{PRF}_p.\mathsf{Setup}(1^\lambda)$.
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) **Compute keys and pads:**
      i. $\widetilde{\mathsf{st}}_1 = p_1 = \mathsf{PRF}_p.\mathsf{Eval}(K_p, 1)$.
      ii. For $j \in [\ell_{\mathcal{F}}]$, $b \in \{0,1\}$, $K_{j,b} = \mathsf{PRF}_2.\mathsf{Setup}(1^\lambda; \mathsf{PRF}_1.\mathsf{Eval}(K, (j, b)))$.
      iii. For $k \in [\ell_{\mathcal{S}}]$, $b \in \{0,1\}$, $\mathsf{sk}'_{1,k,b} = \mathsf{SKE.Setup}(1^\lambda; \mathsf{PRF}_3.\mathsf{Eval}(K', (1, k, b)))$.
   (c) Send $\mathsf{SK}_f = (f, \{K_{j,f[j]}\}_{j\in[\ell_{\mathcal{F}}]}, (\widetilde{\mathsf{st}}_1, \{\mathsf{sk}'_{1,k,\widetilde{\mathsf{st}}_1[k]}\}_{k\in[\ell_{\mathcal{S}}]}))$ to $\mathcal{A}$.
4. **Challenge Message Queries:** For $i = 1, 2, 3, \ldots$
   (a) $\mathcal{A}$ outputs a challenge message $x_i \in \{0,1\}^{\ell_{\mathcal{X}}}$.
   (b) **Compute keys and pads:**
      i. $p_{i+1} = \mathsf{PRF}_p.\mathsf{Eval}(K_p, i+1)$.
      ii. For $j \in [\ell_{\mathcal{F}}]$, $b \in \{0,1\}$, $\mathsf{sk}_{i,j,b} = \mathsf{SKE.Setup}(1^\lambda; \mathsf{PRF}_2.\mathsf{Eval}(K_{j,b}, i))$.
      iii. For $k \in [\ell_{\mathcal{S}}]$, $b \in \{0,1\}$, $\mathsf{sk}'_{i+1,k,b} = \mathsf{SKE.Setup}(1^\lambda; \mathsf{PRF}_3.\mathsf{Eval}(K', (i+1, k, b)))$.
   (c) $\mathsf{st}_1 = 0^{\ell_{\mathcal{S}}}$.
   (d) **Compute** $(y_i, \widetilde{\mathsf{st}}_{i+1})$**:**
      i. $(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$.
      ii. $\widetilde{\mathsf{st}}_{i+1} = \mathsf{st}_{i+1} \oplus p_{i+1}$.
   (e) **Garble circuit:**
      i. ~~Compute $p_i, p_{i+1}, \{\mathsf{sk}'_{i+1,k,b}\}_{k\in[\ell_{\mathcal{S}}],b\in\{0,1\}}$ from K', K_p.~~
      ii. Let $U_i = U[x_i, p_i, p_{i+1}, \{\mathsf{sk}'_{i+1,k,b}\}_{k\in[\ell_{\mathcal{S}}],b\in\{0,1\}}]$ as defined in Figure 8.
      iii. $(\widetilde{U}_i, \{\mathsf{lab}_{i,j,b}\}_{j\in[\ell_{\mathcal{F}}],b\in\{0,1\}}, \{\mathsf{lab}'_{i,k,b}\}_{k\in[\ell_{\mathcal{S}}],b\in\{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, U_i)$.
   (f) **Encrypt function labels:** For $j \in [\ell_{\mathcal{F}}]$, $b \in \{0,1\}$,
      i. ~~Compute $\mathsf{sk}_{i,j,b}$ from K.~~
      ii. $\mathsf{ct}_{i,j,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{i,j,b}, \mathsf{lab}_{i,j,b})$.
   (g) **Encrypt state labels:** For $k \in [\ell_{\mathcal{S}}]$, $b \in \{0,1\}$,
      i. ~~Compute $\mathsf{sk}'_{i,k,b}$ from K'.~~
      ii. $\mathsf{ct}'_{i,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}'_{i,k,b}, \mathsf{lab}'_{i,k,b})$.
   (h) Send $\mathsf{CT}_i = (\widetilde{U}_i, \{\mathsf{ct}_{i,j,b}\}_{j\in[\ell_{\mathcal{F}}],b\in\{0,1\}}, \{\mathsf{ct}'_{i,k,b}\}_{k\in[\ell_{\mathcal{S}}],b\in\{0,1\}})$ to $\mathcal{A}$.
5. **Experiment Outcome:** $\mathcal{A}$ outputs a bit b' which is the output of the experiment.

Lemma 4.3. *For all adversaries* $\mathcal{A}$,

$$\left|\Pr[\mathbf{Hybrid}_0^{\mathcal{A}}(1^\lambda)] - \Pr[\mathbf{Hybrid}_1^{\mathcal{A}}(1^\lambda)]\right| = 0.$$

Proof. The hybrids are identical. ∎

$\mathbf{Hybrid}_2^{\mathcal{A}}(1^\lambda)$: We exchange the randomness generated by K, K', K_p 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:** $K \leftarrow \mathsf{PRF}_1.\mathsf{Setup}(1^\lambda)$, $K' \leftarrow \mathsf{PRF}_3.\mathsf{Setup}(1^\lambda)$, $K_p \leftarrow \mathsf{PRF}_p.\mathsf{Setup}(1^\lambda)$.~~

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) **Compute keys and pads:**
      i. $\widetilde{\mathsf{st}}_1 = p_1 \leftarrow \{0,1\}^{\ell_{\mathcal{S}}}$.
      ii. For $j \in [\ell_{\mathcal{F}}]$, $b \in \{0,1\}$, $K_{j,b} \leftarrow \mathsf{PRF}_2.\mathsf{Setup}(1^\lambda)$.
      iii. For $k \in [\ell_{\mathcal{S}}]$, $b \in \{0,1\}$, $\mathsf{sk}'_{1,k,b} \leftarrow \mathsf{SKE}.\mathsf{Setup}(1^\lambda)$.
   (c) Send $\mathsf{SK}_f = (f, \{K_{j,f[j]}\}_{j \in [\ell_{\mathcal{F}}]}, (\widetilde{\mathsf{st}}_1, \{\mathsf{sk}'_{1,k,\widetilde{\mathsf{st}}_1[k]}\}_{k \in [\ell_{\mathcal{S}}]}))$ to $\mathcal{A}$.

4. **Challenge Message Queries:** For $i = 1, 2, 3, \ldots$
   (a) $\mathcal{A}$ outputs a challenge message $x_i \in \{0,1\}^{\ell_{\mathcal{X}}}$.
   (b) **Compute keys and pads:**
      i. $p_{i+1} \leftarrow \{0,1\}^{\ell_{\mathcal{S}}}$.
      ii. For $j \in [\ell_{\mathcal{F}}]$, $b \in \{0,1\}$, $\mathsf{sk}_{i,j,b} = \mathsf{SKE}.\mathsf{Setup}(1^\lambda; \mathsf{PRF}_2.\mathsf{Eval}(K_{j,b}, i))$.
      iii. For $k \in [\ell_{\mathcal{S}}]$, $b \in \{0,1\}$, $\mathsf{sk}'_{i+1,k,b} \leftarrow \mathsf{SKE}.\mathsf{Setup}(1^\lambda)$.
   (c) $\mathsf{st}_1 = 0^{\ell_{\mathcal{S}}}$.
   (d) **Compute** $(y_i, \widetilde{\mathsf{st}}_{i+1})$**:**
      i. $(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$.
      ii. $\widetilde{\mathsf{st}}_{i+1} = \mathsf{st}_{i+1} \oplus p_{i+1}$.
   (e) **Garble circuit:**
      i. Let $U_i = U[x_i, p_i, p_{i+1}, \{\mathsf{sk}'_{i+1,k,b}\}_{k \in [\ell_{\mathcal{S}}], b \in \{0,1\}}]$ as defined in Figure 8.
      ii. $(\widetilde{U}_i, \{\mathsf{lab}_{i,j,b}\}_{j \in [\ell_{\mathcal{F}}], b \in \{0,1\}}, \{\mathsf{lab}'_{i,k,b}\}_{k \in [\ell_{\mathcal{S}}], b \in \{0,1\}}) \leftarrow \mathsf{GC}.\mathsf{Garble}(1^\lambda, U_i)$.
   (f) **Encrypt function labels:** For $j \in [\ell_{\mathcal{F}}]$, $b \in \{0,1\}$,
      i. $\mathsf{ct}_{i,j,b} \leftarrow \mathsf{SKE}.\mathsf{Enc}(\mathsf{sk}_{i,j,b}, \mathsf{lab}_{i,j,b})$.
   (g) **Encrypt state labels:** For $k \in [\ell_{\mathcal{S}}]$, $b \in \{0,1\}$,
      i. $\mathsf{ct}'_{i,k,b} \leftarrow \mathsf{SKE}.\mathsf{Enc}(\mathsf{sk}'_{i,k,b}, \mathsf{lab}'_{i,k,b})$.
   (h) Send $\mathsf{CT}_i = (\widetilde{U}_i, \{\mathsf{ct}_{i,j,b}\}_{j \in [\ell_{\mathcal{F}}], b \in \{0,1\}}, \{\mathsf{ct}'_{i,k,b}\}_{k \in [\ell_{\mathcal{S}}], b \in \{0,1\}})$ to $\mathcal{A}$.

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

Lemma 4.4. *If* $\mathsf{PRF}_1, \mathsf{PRF}_3$ *and* PRF_p *are secure PRFs, then for all PPT adversaries* $\mathcal{A}$,

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

Proof. We will first show indistinguishability between $\mathbf{Hybrid}_1^{\mathcal{A}}$ and an intermediate hybrid $\mathbf{Hybrid}_{1,1}^{\mathcal{A}}(1^\lambda)$ which is the same as $\mathbf{Hybrid}_1^{\mathcal{A}}$ except that all the outputs of PRF_1 have been replaced by truly random strings (but the outputs of PRF_3 and PRF_p are still pseudorandom values). Suppose for sake of contradiction, that there exists a PPT adversary $\mathcal{A}$ such that

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

We build a PPT adversary $\mathcal{B}$ that breaks the security of PRF_1. $\mathcal{B}$ gets the security parameter from its PRF_1 challenger and provides it to $\mathcal{A}$ who outputs parameters $(1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}})$ to $\mathcal{B}$. $\mathcal{B}$ samples keys $K' \leftarrow \mathsf{PRF}_3.\mathsf{Setup}(1^\lambda)$ and $K_p \leftarrow \mathsf{PRF}_p.\mathsf{Setup}(1^\lambda)$. $\mathcal{B}$ also queries its PRF_1 challenger on values $\{(j,b)\}_{j\in[\ell_{\mathcal{F}}],b\in\{0,1\}}$ and receive values $\{r_{j,b}\}_{j\in[\ell_{\mathcal{F}}],b\in\{0,1\}}$, where the string $r_{j,b}$ is the output obtained on query (j,b). Upon receiving a function query from $\mathcal{A}$ for some function $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$, $\mathcal{B}$ samples PRF_2 keys $K_{j,b} = \mathsf{PRF}_2.\mathsf{Setup}(1^\lambda; r_{j,b})$ for $j \in [\ell_{\mathcal{F}}], b \in \{0,1\}$. $\mathcal{B}$ then performs the rest of the operations as in $\mathbf{Hybrid}_1^{\mathcal{A}}(1^\lambda)$ (while interacting with $\mathcal{A}$) and eventually outputs the output bit b' received from $\mathcal{A}$ as its own output.

Observe that if $\mathcal{B}$'s PRF_1 oracle was a uniform random function R, then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}_{1,1}^{\mathcal{A}}$, and if $\mathcal{B}$'s PRF_1 oracle was $\mathsf{PRF}_1.\mathsf{Eval}(K,\cdot)$ for some PRF_1 key K, then $\mathcal{B}$ emulates $\mathbf{Hybrid}_1^{\mathcal{A}}$. Moreover, $\mathcal{B}$ does not need to know the PRF_1 key K for performing these experiments, as it is not used in any place other than for the function query responses. Therefore, by Equation 1, this means that $\mathcal{B}$ breaks the security of PRF_1 since $\mathcal{B}$ has a non-negligible advantage in distinguishing between the two challenge oracles in the PRF_1 experiment.

Using a similar argument, we can prove that the outputs of PRF_3 and PRF_p can also be replaced with random values, resulting in $\mathbf{Hybrid}_2^{\mathcal{A}}$.

$\square$

$\textbf{Hybrid}_3^{\mathcal{A}}(\lambda)$: 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:**

   (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) **Compute keys and pads:**

      i. $p_1 = \widetilde{\mathsf{st}}_1 \leftarrow \{0,1\}^{\ell_\mathcal{S}}$.

      ii. For $j \in [\ell_\mathcal{F}]$, $b \in \{0,1\}$, $K_{j,b} \leftarrow \mathsf{PRF}_2.\mathsf{Setup}(1^\lambda)$.

      iii. For $k \in [\ell_\mathcal{S}]$, $b \in \{0,1\}$, $\mathsf{sk}'_{1,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.

   (c) Send $\mathsf{SK}_f = (f, \{K_{j,f[j]}\}_{j\in[\ell_\mathcal{F}]}, (\widetilde{\mathsf{st}}_1, \{\mathsf{sk}'_{1,k,\widetilde{\mathsf{st}}_1[k]}\}_{k\in[\ell_\mathcal{S}]}))$ to $\mathcal{A}$.

3. **Challenge Message Queries:** For $i = 1, 2, 3, \ldots$

   (a) $\mathcal{A}$ outputs a challenge message $x_i \in \{0,1\}^{\ell_\mathcal{X}}$.

   (b) **Compute keys and pads:**

      i. $\widetilde{\mathsf{st}}_{i+1} \leftarrow \{0,1\}^{\ell_\mathcal{S}}$.

      ii. For $j \in [\ell_\mathcal{F}]$, $b \in \{0,1\}$, $\mathsf{sk}_{i,j,b} = \mathsf{SKE.Setup}(1^\lambda; \mathsf{PRF}_2.\mathsf{Eval}(K_{j,b}, i))$.

      iii. For $k \in [\ell_\mathcal{S}]$, $b \in \{0,1\}$, $\mathsf{sk}'_{i+1,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.

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

   (d) **Compute** (y_i, p_{i+1})**:**

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

      ii. $p_{i+1} = \mathsf{st}_{i+1} \oplus \widetilde{\mathsf{st}}_{i+1}$.

   (e) **Garble circuit:**

      i. Let $U_i = U[x_i, p_i, p_{i+1}, \{\mathsf{sk}'_{i+1,k,b}\}_{k\in[\ell_\mathcal{S}], b\in\{0,1\}}]$ as defined in Figure 8.

      ii. $(\widetilde{U}_i, \{\mathsf{lab}_{i,j,b}\}_{j\in[\ell_\mathcal{F}], b\in\{0,1\}}, \{\mathsf{lab}'_{i,k,b}\}_{k\in[\ell_\mathcal{S}], b\in\{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, U_i)$.

   (f) **Encrypt function labels:** For $j \in [\ell_\mathcal{F}]$, $b \in \{0,1\}$,

      i. $\mathsf{ct}_{i,j,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{i,j,b}, \mathsf{lab}_{i,j,b})$.

   (g) **Encrypt state labels:** For $k \in [\ell_\mathcal{S}]$, $b \in \{0,1\}$,

      i. $\mathsf{ct}'_{i,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}'_{i,k,b}, \mathsf{lab}'_{i,k,b})$.

   (h) Send $\mathsf{CT}_i = (\widetilde{U}_i, \{\mathsf{ct}_{i,j,b}\}_{j\in[\ell_\mathcal{F}], b\in\{0,1\}}, \{\mathsf{ct}'_{i,k,b}\}_{k\in[\ell_\mathcal{S}], b\in\{0,1\}})$ to $\mathcal{A}$.

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

Lemma 4.5. *For all adversaries* $\mathcal{A}$,

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

Proof. The hybrids are identically distributed since we have just switched the roles of variables p_i and $\widetilde{\mathsf{st}}_i$ which were uniformly distributed random variables conditioned on $\mathsf{st}_i = p_i \oplus \widetilde{\mathsf{st}}_i$. $\square$

$\mathbf{Hybrid}_4^{\mathcal{A}}(\lambda)$: We exchange the randomness generated by $K_{j,1-f[j]}$ 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:**
   (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) **Compute keys and pads:**
      i. $p_1 = \widetilde{\mathsf{st}}_1 \leftarrow \{0,1\}^{\ell_\mathcal{S}}$.
      ii. For $j \in [\ell_\mathcal{F}]$, $K_{j,f[j]} \leftarrow \mathsf{PRF}_2.\mathsf{Setup}(1^\lambda)$.
      iii. For $k \in [\ell_\mathcal{S}]$, $b \in \{0,1\}$, $\mathsf{sk}'_{1,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
   (c) Send $\mathsf{SK}_f = (f, \{K_{j,f[j]}\}_{j\in[\ell_\mathcal{F}]}, (\widetilde{\mathsf{st}}_1, \{\mathsf{sk}'_{1,k,\widetilde{\mathsf{st}}_1[k]}\}_{k\in[\ell_\mathcal{S}]}))$ to $\mathcal{A}$.
3. **Challenge Message Queries:** For $i = 1, 2, 3, \ldots$
   (a) $\mathcal{A}$ outputs a challenge message $x_i \in \{0,1\}^{\ell_\mathcal{X}}$.
   (b) **Compute keys and pads:**
      i. $\widetilde{\mathsf{st}}_{i+1} \leftarrow \{0,1\}^{\ell_\mathcal{S}}$.
      ii. For $j \in [\ell_\mathcal{F}]$, $b \in \{0,1\}$,
         A. If $b = f[j]$, $\mathsf{sk}_{i,j,b} = \mathsf{SKE.Setup}(1^\lambda; \mathsf{PRF}_2.\mathsf{Eval}(K_{j,b}, i))$.
         B. If $b \neq f[j]$, $\mathsf{sk}_{i,j,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
      iii. For $k \in [\ell_\mathcal{S}]$, $b \in \{0,1\}$, $\mathsf{sk}'_{i+1,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
   (c) $\mathsf{st}_1 = 0^{\ell_\mathcal{S}}$.
   (d) **Compute** (y_i, p_{i+1})**:**
      i. $(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$.
      ii. $p_{i+1} = \mathsf{st}_{i+1} \oplus \widetilde{\mathsf{st}}_{i+1}$.
   (e) **Garble circuit:**
      i. Let $U_i = U[x_i, p_i, p_{i+1}, \{\mathsf{sk}'_{i+1,k,b}\}_{k\in[\ell_\mathcal{S}], b\in\{0,1\}}]$ as defined in Figure 8.
      ii. $(\widetilde{U}_i, \{\mathsf{lab}_{i,j,b}\}_{j\in[\ell_\mathcal{F}], b\in\{0,1\}}, \{\mathsf{lab}'_{i,k,b}\}_{k\in[\ell_\mathcal{S}], b\in\{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, U_i)$.
   (f) **Encrypt function labels:** For $j \in [\ell_\mathcal{F}]$, $b \in \{0,1\}$,
      i. $\mathsf{ct}_{i,j,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{i,j,b}, \mathsf{lab}_{i,j,b})$.
   (g) **Encrypt state labels:** For $k \in [\ell_\mathcal{S}]$, $b \in \{0,1\}$,
      i. $\mathsf{ct}'_{i,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}'_{i,k,b}, \mathsf{lab}'_{i,k,b})$.
   (h) Send $\mathsf{CT}_i = (\widetilde{U}_i, \{\mathsf{ct}_{i,j,b}\}_{j\in[\ell_\mathcal{F}], b\in\{0,1\}}, \{\mathsf{ct}'_{i,k,b}\}_{k\in[\ell_\mathcal{S}], b\in\{0,1\}})$ to $\mathcal{A}$.
4. **Experiment Outcome:** $\mathcal{A}$ outputs a bit b' which is the output of the experiment.

Lemma 4.6. *If* PRF_2 *is a secure PRF, then for all PPT adversaries* $\mathcal{A}$,

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

Proof. For $J \in [\ell_{\mathcal{F}}]$, we define sub-hybrid $\mathbf{Hybrid}_{3,J}^{\mathcal{A}}(1^\lambda)$ to be the same as $\mathbf{Hybrid}_3$ except that for $j \le J$, we do not compute $K_{j,1-f[j]}$ during the hybrid and thus sample the corresponding keys $\mathsf{sk}_{i,j,1-f[j]} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$ for each i using uniform randomness. In other words, with regards to PRF_2, sub-hybrid $\mathbf{Hybrid}_{3,J}^{\mathcal{A}}(1^\lambda)$ behaves identically to $\mathbf{Hybrid}_4^{\mathcal{A}}(1^\lambda)$ for $j \le J$, and $\mathbf{Hybrid}_{3,J}^{\mathcal{A}}(1^\lambda)$ behaves identically to $\mathbf{Hybrid}_3^{\mathcal{A}}(1^\lambda)$ for $j > J$. Observe that $\mathbf{Hybrid}_{3,0}^{\mathcal{A}} = \mathbf{Hybrid}_3^{\mathcal{A}}$ and $\mathbf{Hybrid}_{3,\ell_{\mathcal{F}}}^{\mathcal{A}} = \mathbf{Hybrid}_4^{\mathcal{A}}$.

We now show that for all $J \in [\ell_{\mathcal{F}}]$, $\mathbf{Hybrid}_{3,J-1}^{\mathcal{A}}$ and $\mathbf{Hybrid}_{3,J}^{\mathcal{A}}$ are indistinguishable. This proves our lemma. Suppose for sake of contradiction, that there exists a PPT adversary $\mathcal{A}$ and an index $J \in [\ell_{\mathcal{F}}]$ such that

$$\left| \Pr[\mathbf{Hybrid}_{3,J-1}^{\mathcal{A}}(1^\lambda)] - \Pr[\mathbf{Hybrid}_{3,J}^{\mathcal{A}}(1^\lambda)] \right| > \mathsf{negl}(\lambda) \tag{2}$$

We build a PPT adversary $\mathcal{B}$ that breaks the security of PRF_2. $\mathcal{B}$ follows the steps of $\mathbf{Hybrid}_{3,J}^{\mathcal{A}}$ by interacting with $\mathcal{A}$ except that on each challenge message query x_i, $\mathcal{B}$ computes $\mathsf{sk}_{i,J,1-f[J]} \leftarrow \mathsf{SKE.Setup}(1^\lambda, r_{i,J})$ where $r_{i,J}$ is the output of $\mathcal{B}$'s PRF_2 oracle on input i. $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

Observe that if $\mathcal{B}$'s PRF_2 oracle was a uniform random function R, then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}_{3,J-1}^{\mathcal{A}}$, and if $\mathcal{B}$'s PRF_2 oracle was $\mathsf{PRF}_2.\mathsf{Eval}(K_{J,1-f[J]}, \cdot)$ for some PRF_2 key $K_{J,1-f[J]}$, then $\mathcal{B}$ emulates $\mathbf{Hybrid}_{3,J}^{\mathcal{A}}$. Moreover, $\mathcal{B}$ does not need to know the PRF_2 key $K_{J,1-f[J]}$ for performing these experiments. Therefore, by Equation 2, this means that $\mathcal{B}$ breaks the security of PRF_2 since $\mathcal{B}$ has a non-negligible advantage in distinguishing between the two challenge oracles in the PRF_2 experiment.

□

$\mathbf{Hybrid}_5^{\mathcal{A}}(\lambda)$: We replace the ciphertexts $\mathsf{ct}_{i,j,1-f[j]}$ with encryptions of $\perp$. This removes the input labels which don't correspond to f from the adversary's view.

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:**

   (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) **Compute keys and pads:**

      i. $p_1 = \widetilde{\mathsf{st}}_1 \leftarrow \{0,1\}^{\ell_\mathcal{S}}$.

      ii. For $j \in [\ell_\mathcal{F}]$, $K_{j,f[j]} \leftarrow \mathsf{PRF}_2.\mathsf{Setup}(1^\lambda)$.

      iii. For $k \in [\ell_\mathcal{S}]$, $b \in \{0,1\}$, $\mathsf{sk}'_{1,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.

   (c) Send $\mathsf{SK}_f = (f, \{K_{j,f[j]}\}_{j\in[\ell_\mathcal{F}]}, (\widetilde{\mathsf{st}}_1, \{\mathsf{sk}'_{1,k,\widetilde{\mathsf{st}}_1[k]}\}_{k\in[\ell_\mathcal{S}]}))$ to $\mathcal{A}$.

3. **Challenge Message Queries:** For $i = 1, 2, 3, \ldots$

   (a) $\mathcal{A}$ outputs a challenge message $x_i \in \{0,1\}^{\ell_\mathcal{X}}$.

   (b) **Compute keys and pads:**

      i. $\widetilde{\mathsf{st}}_{i+1} \leftarrow \{0,1\}^{\ell_\mathcal{S}}$.

      ii. For $j \in [\ell_\mathcal{F}]$, $b \in \{0,1\}$,

         A. If $b = f[j]$, $\mathsf{sk}_{i,j,b} = \mathsf{SKE.Setup}(1^\lambda; \mathsf{PRF}_2.\mathsf{Eval}(K_{j,b}, i))$.

         B. If $b \neq f[j]$, $\mathsf{sk}_{i,j,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.

      iii. For $k \in [\ell_\mathcal{S}]$, $b \in \{0,1\}$, $\mathsf{sk}'_{i+1,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.

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

   (d) **Compute** (y_i, p_{i+1})**:**

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

      ii. $p_{i+1} = \mathsf{st}_{i+1} \oplus \widetilde{\mathsf{st}}_{i+1}$.

   (e) **Garble circuit:**

      i. Let $U_i = U[x_i, p_i, p_{i+1}, \{\mathsf{sk}'_{i+1,k,b}\}_{k\in[\ell_\mathcal{S}], b\in\{0,1\}}]$ as defined in Figure 8.

      ii. $(\widetilde{U}_i, \{\mathsf{lab}_{i,j,b}\}_{j\in[\ell_\mathcal{F}], b\in\{0,1\}}, \{\mathsf{lab}'_{i,k,b}\}_{k\in[\ell_\mathcal{S}], b\in\{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, U_i)$.

   (f) **Encrypt function labels:** For $j \in [\ell_\mathcal{F}]$, $b \in \{0,1\}$,

      i. If $b = f[j]$, $\mathsf{ct}_{i,j,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{i,j,b}, \mathsf{lab}_{i,j,b})$.

      ii. If $b \neq f[j]$, $\mathsf{ct}_{i,j,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{i,j,b}, \perp)$.

   (g) **Encrypt state labels:** For $k \in [\ell_\mathcal{S}]$, $b \in \{0,1\}$,

      i. $\mathsf{ct}'_{i,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}'_{i,k,b}, \mathsf{lab}'_{i,k,b})$.

   (h) Send $\mathsf{CT}_i = (\widetilde{U}_i, \{\mathsf{ct}_{i,j,b}\}_{j\in[\ell_\mathcal{F}], b\in\{0,1\}}, \{\mathsf{ct}'_{i,k,b}\}_{k\in[\ell_\mathcal{S}], b\in\{0,1\}})$ to $\mathcal{A}$.

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

Lemma 4.7. *If* SKE *is a secure secret-key encryption scheme, then for all PPT adversaries* $\mathcal{A}$*,*

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

Proof. For the sake of contradiction, assume that there exists a PPT adversary $\mathcal{A}$ such that

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \left|\Pr[\mathbf{Hybrid}_4^{\mathcal{A}}(1^\lambda)] - \Pr[\mathbf{Hybrid}_5^{\mathcal{A}}(1^\lambda)]\right| \tag{3}$$

is a non-negligible function in λ. Let n be the number of challenge message queries requested by $\mathcal{A}$. For each $I \in [n]$ and $J \in \{0, \dots, \ell_{\mathcal{F}}\}$, we define sub-hybrid $\mathbf{Hybrid}_{4,I,J}^{\mathcal{A}}$ to be the same as $\mathbf{Hybrid}_4^{\mathcal{A}}$ except that for indices (i,j) where either $i < I$ or ($i = I$ and $j \le J$), $\mathbf{Hybrid}_{4,I,J}^{\mathcal{A}}$ computes $\mathsf{ct}_{i,j,1-f[j]}$ as an encryption of $\perp$ rather than an encryption of $\mathsf{lab}_{i,j,1-f[j]}$.

For each $I \in [n], J \in [\ell_{\mathcal{F}}]$, let

$$\mathsf{Adv}_{\mathcal{A}}^{I,J}(\lambda) = \Pr[\mathbf{Hybrid}_{4,I,J-1}^{\mathcal{A}}(1^\lambda)] - \Pr[\mathbf{Hybrid}_{4,I,J}^{\mathcal{A}}(1^\lambda)]$$

Then, by Equation 3, since

- $\mathbf{Hybrid}_4^{\mathcal{A}}(1^\lambda) = \mathbf{Hybrid}_{4,1,0}^{\mathcal{A}}(1^\lambda)$,
- $\mathbf{Hybrid}_5^{\mathcal{A}}(1^\lambda) = \mathbf{Hybrid}_{4,n,\ell_{\mathcal{F}}}^{\mathcal{A}}(1^\lambda)$,
- for all $i \in [n]$, $\mathbf{Hybrid}_{4,I-1,\ell_{\mathcal{F}}}^{\mathcal{A}} = \mathbf{Hybrid}_{4,I,0}^{\mathcal{A}}$,

there must exist an $I \in [n]$ and $J \in [\ell_{\mathcal{F}}]$ such that $\mathsf{Adv}_{\mathcal{A}}^{I,J}(\lambda)$ is a non-negligible function in λ. Let (I, J) be such values. We build a PPT adversary $\mathcal{B}$ that breaks the security of SKE.

$\mathcal{B}$ receives the security parameter from its SKE challenger and provides it to $\mathcal{A}$ who outputs parameters $(1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}})$ to $\mathcal{B}$. When $\mathcal{A}$ outputs a challenge function query f, $\mathcal{B}$ behaves identically to $\mathbf{Hybrid}_{4,I,J-1}^{\mathcal{A}}(1^\lambda)$. When $\mathcal{A}$ outputs a challenge message query x_i, $\mathcal{B}$ behaves identically to $\mathbf{Hybrid}_{4,I,J-1}^{\mathcal{A}}(1^\lambda)$ except that if $i = I$, rather than computing $\mathsf{ct}_{I,J,1-f[j]}$ as in $\mathbf{Hybrid}_{4,I,J-1}^{\mathcal{A}}(1^\lambda)$, $\mathcal{B}$ sends challenge message pair $(\mathsf{lab}_{I,J,1-f[J]}, \perp)$ to its SKE challenger, receives back ct^* from its SKE challenger, and sets $\mathsf{ct}_{I,J,1-f[J]} = \mathsf{ct}^*$. At the end of the experiment, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

Observe that if ct^* was an encryption of $\mathsf{lab}_{I,J,1-f[J]}$, then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}_{4,I,J-1}^{\mathcal{A}}$, and if ct^* was an encryption of $\perp$, then $\mathcal{B}$ emulates $\mathbf{Hybrid}_{4,I,J}^{\mathcal{A}}$. $\mathcal{B}$ is a valid SKE adversary because the secret key $\mathsf{sk}_{I,J,1-f[J]}$ is not needed by $\mathcal{B}$. Therefore, $\mathcal{B}$ has non-negligible advantage $\mathsf{Adv}_{\mathcal{A}}^{I,J}$ in breaking its SKE game, contradicting the security of SKE.

□

We now go through the following hybrids for $\alpha \in [\mathsf{Bound}_{\mathcal{A}}]$ where $\mathsf{Bound}_{\mathcal{A}}$ is a bound on the runtime of $\mathcal{A}$, and thus an implicit bound on the number of challenge message queries made by the adversary. On iteration α, the goal is to switch to a hybrid where we simulate the α^{th} garbled circuit.

Hybrid$^{\mathcal{A}}_{6,\alpha,0}(\lambda)$: We replace the ciphertexts $\mathsf{ct}'_{\alpha,k,1-\widetilde{\mathsf{st}}[\alpha]}$ with encryptions of $\perp$. This removes the input labels which don't correspond to $\widetilde{\mathsf{st}}_\alpha$ from the adversary's view.

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:**

   (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) **Compute keys and pads:**

      i. $p_1 = \widetilde{\mathsf{st}}_1 \leftarrow \{0,1\}^{\ell_{\mathcal{S}}}$.

      ii. For $j \in [\ell_{\mathcal{F}}]$, $K_{j,f[j]} \leftarrow \mathsf{PRF}_2.\mathsf{Setup}(1^\lambda)$.

      iii. For $k \in [\ell_{\mathcal{S}}]$, $b \in \{0,1\}$, $\mathsf{sk}'_{1,k,b} \leftarrow \mathsf{SKE}.\mathsf{Setup}(1^\lambda)$.

   (c) Send $\mathsf{SK}_f = (f, \{K_{j,f[j]}\}_{j\in[\ell_{\mathcal{F}}]}, (\widetilde{\mathsf{st}}_1, \{\mathsf{sk}'_{1,k,\widetilde{\mathsf{st}}_1[k]}\}_{k\in[\ell_{\mathcal{S}}]}))$ to $\mathcal{A}$.

3. **Challenge Message Queries:** For $i = 1, 2, 3, \ldots$

   (a) $\mathcal{A}$ outputs a challenge message $x_i \in \{0,1\}^{\ell_{\mathcal{X}}}$.

   (b) **Compute keys and pads:**

      i. $\widetilde{\mathsf{st}}_{i+1} \leftarrow \{0,1\}^{\ell_{\mathcal{S}}}$.

      ii. For $j \in [\ell_{\mathcal{F}}]$, $b \in \{0,1\}$,

         A. If $b = f[j]$, $\mathsf{sk}_{i,j,b} = \mathsf{SKE}.\mathsf{Setup}(1^\lambda; \mathsf{PRF}_2.\mathsf{Eval}(K_{j,b}, i))$.

         B. If $b \neq f[j]$, $\mathsf{sk}_{i,j,b} \leftarrow \mathsf{SKE}.\mathsf{Setup}(1^\lambda)$.

      iii. For $k \in [\ell_{\mathcal{S}}]$, $b \in \{0,1\}$, $\mathsf{sk}'_{i+1,k,b} \leftarrow \mathsf{SKE}.\mathsf{Setup}(1^\lambda)$.

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

   (d) **Compute** (y_i, p_{i+1})**:**

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

      ii. $p_{i+1} = \mathsf{st}_{i+1} \oplus \widetilde{\mathsf{st}}_{i+1}$.

   (e) **Garble circuit:**

      i. If $i \geq \alpha$,

         A. Let $U_i = U[x_i, p_i, p_{i+1}, \{\mathsf{sk}'_{i+1,k,b}\}_{k\in[\ell_{\mathcal{S}}],b\in\{0,1\}}]$ as defined in Figure 8.

         B. $(\widetilde{U}_i, \{\mathsf{lab}_{i,j,b}\}_{j\in[\ell_{\mathcal{F}}],b\in\{0,1\}}, \{\mathsf{lab}'_{i,k,b}\}_{k\in[\ell_{\mathcal{S}}],b\in\{0,1\}}) \leftarrow \mathsf{GC}.\mathsf{Garble}(1^\lambda, U_i)$.

      ii. If $i < \alpha$,

         A. $(\widetilde{U}_i, \{\mathsf{lab}_{i,j,f[j]}\}_{j\in[\ell_{\mathcal{F}}]}, \{\mathsf{lab}'_{i,k,\widetilde{\mathsf{st}}_i[k]}\}_{k\in[\ell_{\mathcal{S}}]})$
         $\leftarrow \mathsf{GC}.\mathsf{Sim}(1^\lambda, 1^{\ell_U}, (f, \widetilde{\mathsf{st}}_i), (y_i, \{\mathsf{sk}'_{i+1,k,\widetilde{\mathsf{st}}_{i+1}[k]}\}_{k\in[\ell_{\mathcal{S}}]}))$

   (f) **Encrypt function labels:** For $j \in [\ell_{\mathcal{F}}]$, $b \in \{0,1\}$,

      i. If $b = f[j]$, $\mathsf{ct}_{i,j,b} \leftarrow \mathsf{SKE}.\mathsf{Enc}(\mathsf{sk}_{i,j,b}, \mathsf{lab}_{i,j,b})$.

      ii. If $b \neq f[j]$, $\mathsf{ct}_{i,j,b} \leftarrow \mathsf{SKE}.\mathsf{Enc}(\mathsf{sk}_{i,j,b}, \perp)$.

   (g) **Encrypt state labels:** For $k \in [\ell_{\mathcal{S}}]$, $b \in \{0,1\}$,

i. If $i > \alpha$ or $b = \widetilde{\mathsf{st}}_i[k]$, $\mathsf{ct}'_{i,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}'_{i,k,b}, \mathsf{lab}'_{i,k,b})$.

ii. If $i \leq \alpha$ and $b \neq \widetilde{\mathsf{st}}_i[k]$, $\mathsf{ct}'_{i,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}'_{i,k,b}, \perp)$.

(h) Send $\mathsf{CT}_i = (\widetilde{U}_i, \{\mathsf{ct}_{i,j,b}\}_{j\in[\ell_{\mathcal{F}}],b\in\{0,1\}}, \{\mathsf{ct}'_{i,k,b}\}_{k\in[\ell_{\mathcal{S}}],b\in\{0,1\}})$ to $\mathcal{A}$.

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

Lemma 4.8. *If* SKE *is a secure secret-key encryption scheme, then for all PPT adversaries* $\mathcal{A}$,

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

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

$$\left|\Pr[\mathbf{Hybrid}_5^{\mathcal{A}}(1^\lambda)] - \Pr[\mathbf{Hybrid}_{6,1,0}^{\mathcal{A}}(1^\lambda)]\right| > \mathsf{negl}(\lambda) \tag{4}$$

We build a PPT adversary $\mathcal{B}$ that breaks the security of SKE. $\mathcal{B}$ receives the security parameter from its SKE challenger and provides it to $\mathcal{A}$ who outputs parameters $(1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{S}}}, 1^{\ell_{\mathcal{X}}}, 1^{\ell_{\mathcal{Y}}})$ to $\mathcal{B}$. When $\mathcal{A}$ outputs a challenge function query f, $\mathcal{B}$ behaves identically to $\mathbf{Hybrid}_5^{\mathcal{A}}(1^\lambda)$ except that $\mathcal{B}$ does not compute $\mathsf{sk}'_{1,k,1-\widetilde{\mathsf{st}}_1[k]}$. When $\mathcal{A}$ outputs a challenge message query x_i, $\mathcal{B}$ behaves identically to $\mathbf{Hybrid}_5^{\mathcal{A}}(1^\lambda)$ except that if $i = 1$, rather than computing $\mathsf{ct}'_{1,k,1-\widetilde{\mathsf{st}}_1[k]}$ as in $\mathbf{Hybrid}_5^{\mathcal{A}}(1^\lambda)$, $\mathcal{B}$ sends challenge message pair $(\mathsf{lab}_{1,k,1-\widetilde{\mathsf{st}}_1[k]}, \perp)$ to its SKE challenger, receives back ct^* from its SKE challenger, and sets $\mathsf{ct}_{1,k,1-\widetilde{\mathsf{st}}[k]} = \mathsf{ct}^*$. At the end of the experiment, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

Observe that if ct^* was an encryption of $\mathsf{lab}_{1,k,1-\widetilde{\mathsf{st}}_1[k]}$, then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}_5^{\mathcal{A}}$, and if ct^* was an encryption of $\perp$, then $\mathcal{B}$ emulates $\mathbf{Hybrid}_{6,1,0}^{\mathcal{A}}$. Moreover, $\mathcal{B}$ does not need to know the SKE key $\mathsf{sk}_{1,k,1-\widetilde{\mathsf{st}}_1[k]}$ for performing these experiments. Therefore, by Equation 4, this means that $\mathcal{B}$ breaks the security of SKE since $\mathcal{B}$ can distinguish between the two SKE ciphertexts with non-negligible probability.

□

$\mathbf{Hybrid}^{\mathcal{A}}_{6,\alpha,1}(\lambda)$: We simulate the garbled circuit for U_α.

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:**
   (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) **Compute keys and pads:**
      i. $p_1 = \widetilde{\mathsf{st}}_1 \leftarrow \{0,1\}^{\ell_\mathcal{S}}$.
      ii. For $j \in [\ell_\mathcal{F}]$, $K_{j,f[j]} \leftarrow \mathsf{PRF}_2.\mathsf{Setup}(1^\lambda)$.
      iii. For $k \in [\ell_\mathcal{S}]$, $b \in \{0,1\}$, $\mathsf{sk}'_{1,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
   (c) Send $\mathsf{SK}_f = (f, \{K_{j,f[j]}\}_{j\in[\ell_\mathcal{F}]}, (\widetilde{\mathsf{st}}_1, \{\mathsf{sk}'_{1,k,\widetilde{\mathsf{st}}_1[k]}\}_{k\in[\ell_\mathcal{S}]}))$ to $\mathcal{A}$.
3. **Challenge Message Queries:** For $i = 1, 2, 3, \ldots$
   (a) $\mathcal{A}$ outputs a challenge message $x_i \in \{0,1\}^{\ell_\mathcal{X}}$.
   (b) **Compute keys and pads:**
      i. $\widetilde{\mathsf{st}}_{i+1} \leftarrow \{0,1\}^{\ell_\mathcal{S}}$.
      ii. For $j \in [\ell_\mathcal{F}]$, $b \in \{0,1\}$,
         A. If $b = f[j]$, $\mathsf{sk}_{i,j,b} = \mathsf{SKE.Setup}(1^\lambda; \mathsf{PRF}_2.\mathsf{Eval}(K_{j,b}, i))$.
         B. If $b \neq f[j]$, $\mathsf{sk}_{i,j,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
      iii. For $k \in [\ell_\mathcal{S}]$, $b \in \{0,1\}$, $\mathsf{sk}'_{i+1,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
   (c) $\mathsf{st}_1 = 0^{\ell_\mathcal{S}}$.
   (d) **Compute** (y_i, p_{i+1})**:**
      i. $(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$.
      ii. $p_{i+1} = \mathsf{st}_{i+1} \oplus \widetilde{\mathsf{st}}_{i+1}$.
   (e) **Garble circuit:**
      i. If $i > \alpha$,
         A. Let $U_i = U[x_i, p_i, p_{i+1}, \{\mathsf{sk}'_{i+1,k,b}\}_{k\in[\ell_\mathcal{S}], b\in\{0,1\}}]$ as defined in Figure 8.
         B. $(\widetilde{U}_i, \{\mathsf{lab}_{i,j,b}\}_{j\in[\ell_\mathcal{F}], b\in\{0,1\}}, \{\mathsf{lab}'_{i,k,b}\}_{k\in[\ell_\mathcal{S}], b\in\{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, U_i)$.
      ii. If $i \leq \alpha$,
         A. $(\widetilde{U}_i, \{\mathsf{lab}_{i,j,f[j]}\}_{j\in[\ell_\mathcal{F}]}, \{\mathsf{lab}'_{i,k,\widetilde{\mathsf{st}}_i[k]}\}_{k\in[\ell_\mathcal{S}]})$
         $\leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{\ell_U}, (f, \widetilde{\mathsf{st}}_i), (y_i, (\widetilde{\mathsf{st}}_{i+1}, \{\mathsf{sk}'_{i+1,k,\widetilde{\mathsf{st}}_{i+1}[k]}\}_{k\in[\ell_\mathcal{S}]})))$.
   (f) **Encrypt function labels:** For $j \in [\ell_\mathcal{F}]$, $b \in \{0,1\}$,
      i. If $b = f[j]$, $\mathsf{ct}_{i,j,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{i,j,b}, \mathsf{lab}_{i,j,b})$.
      ii. If $b \neq f[j]$, $\mathsf{ct}_{i,j,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{i,j,b}, \perp)$.
   (g) **Encrypt state labels:** For $k \in [\ell_\mathcal{S}]$, $b \in \{0,1\}$,
      i. If $i > \alpha$ or $b = \widetilde{\mathsf{st}}_i[k]$, $\mathsf{ct}'_{i,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}'_{i,k,b}, \mathsf{lab}'_{i,k,b})$.
      ii. If $i \leq \alpha$ and $b \neq \widetilde{\mathsf{st}}_i[k]$, $\mathsf{ct}'_{i,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}'_{i,k,b}, \perp)$.
   (h) Send $\mathsf{CT}_i = (\widetilde{U}_i, \{\mathsf{ct}_{i,j,b}\}_{j\in[\ell_\mathcal{F}], b\in\{0,1\}}, \{\mathsf{ct}'_{i,k,b}\}_{k\in[\ell_\mathcal{S}], b\in\{0,1\}})$ to $\mathcal{A}$.
4. **Experiment Outcome:** $\mathcal{A}$ outputs a bit b' which is the output of the experiment.

Lemma 4.9. *If* GC *is a secure garbling scheme, then for all* $\alpha \in \mathbb{N}$ *and all PPT adversaries* $\mathcal{A}$*,*

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,\alpha,0}(1^\lambda)] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,\alpha,1}(1^\lambda)]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Suppose for sake of contradiction, that there exists a PPT adversary $\mathcal{A}$ and an index $\alpha \in \mathbb{N}$ such that

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,\alpha,0}(1^\lambda)] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,\alpha,1}(1^\lambda)]\right| > \mathsf{negl}(\lambda) \tag{5}$$

We build a PPT adversary $\mathcal{B}$ that breaks the security of GC. $\mathcal{B}$ receives the security parameter from its GC challenger and provides it to $\mathcal{A}$ who outputs parameters $(1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{S}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}})$ to $\mathcal{B}$. When $\mathcal{A}$ outputs a challenge function query f, $\mathcal{B}$ behaves identically to $\mathbf{Hybrid}^{\mathcal{A}}_{6,\alpha,0}(1^\lambda)$. When $\mathcal{A}$ outputs a challenge message query x_i where $i \neq \alpha$, $\mathcal{B}$ behaves identically to $\mathbf{Hybrid}^{\mathcal{A}}_{6,\alpha,0}(1^\lambda)$. When $\mathcal{A}$ outputs the α^{th} message query x_α, $\mathcal{B}$ acts similarly to $\mathbf{Hybrid}^{\mathcal{A}}_{6,\alpha,0}(1^\lambda)$ except that rather than computing $(\widetilde{U}_\alpha, \{\mathsf{lab}_{\alpha,j,b}\}_{j\in[\ell_\mathcal{F}],b\in\{0,1\}}, \{\mathsf{lab}'_{\alpha,k,b}\}_{k\in[\ell_\mathcal{S}],b\in\{0,1\}}\})$ as in $\mathbf{Hybrid}^{\mathcal{A}}_{6,\alpha,0}(1^\lambda)$, $\mathcal{B}$ first computes $(U_\alpha, (f, \widetilde{\mathsf{st}}_\alpha))$ as in as $\mathbf{Hybrid}^{\mathcal{A}}_{6,\alpha,0}(1^\lambda)$, sends $(U_\alpha, (f, \widetilde{\mathsf{st}}_\alpha))$ to its GC challenger, and sets $(\widetilde{U}_\alpha, \{\mathsf{lab}_{\alpha,j,f[j]}\}_{j\in[\ell_\mathcal{F}]}, \{\mathsf{lab}'_{\alpha,k,\widetilde{\mathsf{st}}[k]}\}_{k\in[\ell_\mathcal{S}]}\})$ equal to the values output by its GC challenger. Observe that the missing input labels $\{\mathsf{lab}_{\alpha,j,1-f[j]}\}_{j\in[\ell_\mathcal{F}]}, \{\mathsf{lab}'_{\alpha,k,1-\widetilde{\mathsf{st}}[k]}\}_{k\in[\ell_\mathcal{S}]}\}$ which are *not* output by the GC challenger are not needed by $\mathcal{B}$ in these hybrids. At the end of the experiment, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

Observe that if the GC challenger generates the garbled circuits and input labels honestly, then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}^{\mathcal{A}}_{6,\alpha,0}$, and if the GC challenger simluates these values, then $\mathcal{B}$ emulates $\mathbf{Hybrid}^{\mathcal{A}}_{6,\alpha,1}$. Thus, by Equation 5, $\mathcal{B}$ breaks the security of GC since $\mathcal{B}$ has a non-negligible advantage in distinguishing between a real garbling and a simulated one.

□

Lemma 4.10. *If* SKE *is a secure secret-key encryption scheme, then for all* $\alpha \in \mathbb{N}$ *and all PPT adversaries* $\mathcal{A}$*,*

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,\alpha,1}(1^\lambda)] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,\alpha+1,0}(1^\lambda)]\right| \leq \mathsf{negl}(\lambda).$$

Proof. This proof follows by a straightforward reduction to the security of SKE using a proof similar to Lemma 4.8. In particular, for $k \in [\ell_\mathcal{S}]$, we can use the security of SKE to change ciphertexts $\mathsf{ct}'_{\alpha+1,k,1-\widetilde{\mathsf{st}}_{\alpha+1}[k]}$ from encryptions of $\mathsf{lab}'_{\alpha+1,k,1-\widetilde{\mathsf{st}}_{\alpha+1}[k]}$ to encryptions of $\perp$. This can be argued since the secret keys $\mathsf{sk}'_{\alpha+1,k,1-\widetilde{\mathsf{st}}_{\alpha+1}[k]}$ are no longer used anywhere in these hybrids because the garbled circuit for U_α is now being simulated. For brevity, we omit further details. □

$\mathbf{Hybrid}_7^{\mathcal{A}}(\lambda)$: This is the idea world experiment written using an explicit simulator Sim. This hybrid is identical to $\mathbf{Hybrid}^{\mathcal{A}}_{6,\mathsf{Bound}_{\mathcal{A}},1}$ where $\mathsf{Bound}_{\mathcal{A}}$ is a bound on the runtime of $\mathcal{A}$, and thus an implicit bound on the number of message queries $\mathcal{A}$ will make.

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:**
   (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) **Simulated Function Key:** Sim receives f and computes the following:
      i. **Compute keys and pads:**
         A. $p_1 = \widetilde{\mathsf{st}}_1 \leftarrow \{0,1\}^{\ell_{\mathcal{S}}}$.
         B. For $j \in [\ell_{\mathcal{F}}]$, $K_{j,f[j]} \leftarrow \mathsf{PRF}_2.\mathsf{Setup}(1^\lambda)$.
         C. For $k \in [\ell_{\mathcal{S}}]$, $b \in \{0,1\}$, $\mathsf{sk}'_{1,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
      ii. Send $\mathsf{SK}_f = (f, \{K_{j,f[j]}\}_{j\in[\ell_{\mathcal{F}}]}, (\widetilde{\mathsf{st}}_1, \{\mathsf{sk}'_{1,k,\widetilde{\mathsf{st}}_1[k]}\}_{k\in[\ell_{\mathcal{S}}]}))$ to $\mathcal{A}$.
3. **Challenge Message Queries:** For $i = 1, 2, 3, \ldots$
   (a) $\mathcal{A}$ outputs a challenge message $x_i \in \{0,1\}^{\ell_{\mathcal{X}}}$.
   (b) **Compute Output Value**: $(y_i, \mathsf{st}_{i+1}) = f(x_i, \mathsf{st}_i)$ where $\mathsf{st}_1 = 0^{\ell_{\mathcal{S}}}$.
   (c) **Simulated Ciphertext:** Sim receives y_i and computes the following:
      i. $\widetilde{\mathsf{st}}_{i+1} \leftarrow \{0,1\}^{\ell_{\mathcal{S}}}$.
      ii. For $j \in [\ell_{\mathcal{F}}]$, $b \in \{0,1\}$,
         A. If $b = f[j]$, $\mathsf{sk}_{i,j,b} = \mathsf{SKE.Setup}(1^\lambda; \mathsf{PRF}_2.\mathsf{Eval}(K_{j,b}, i))$.
         B. If $b \neq f[j]$, $\mathsf{sk}_{i,j,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
      iii. For $k \in [\ell_{\mathcal{S}}]$, $b \in \{0,1\}$, $\mathsf{sk}'_{i+1,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
      iv. **Garble circuit:**
         A. $(\widetilde{U}_i, \{\mathsf{lab}_{i,j,f[j]}\}_{j\in[\ell_{\mathcal{F}}]}, \{\mathsf{lab}'_{i,k,\widetilde{\mathsf{st}}_i[k]}\}_{k\in[\ell_{\mathcal{S}}]})$
         $\leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{\ell_U}, (f, \widetilde{\mathsf{st}}_i), (y_i, \{\mathsf{sk}'_{i+1,k,\widetilde{\mathsf{st}}_{i+1}[k]}\}_{k\in[\ell_{\mathcal{S}}]}))$.
      v. **Encrypt function labels:** For $j \in [\ell_{\mathcal{F}}]$, $b \in \{0,1\}$,
         A. If $b = f[j]$, $\mathsf{ct}_{i,j,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{i,j,b}, \mathsf{lab}_{i,j,b})$.
         B. If $b \neq f[j]$, $\mathsf{ct}_{i,j,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{i,j,b}, \perp)$.
      vi. **Encrypt state labels:** For $k \in [\ell_{\mathcal{S}}]$, $b \in \{0,1\}$,
         A. If $b = \widetilde{\mathsf{st}}_i[k]$, $\mathsf{ct}'_{i,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}'_{i,k,b}, \mathsf{lab}'_{i,k,b})$.
         B. If $b \neq \widetilde{\mathsf{st}}_i[k]$, $\mathsf{ct}'_{i,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}'_{i,k,b}, \perp)$.
      vii. Send $\mathsf{CT}_i = (\widetilde{U}_i, \{\mathsf{ct}_{i,j,b}\}_{j\in[\ell_{\mathcal{F}}], b\in\{0,1\}}, \{\mathsf{ct}'_{i,k,b}\}_{k\in[\ell_{\mathcal{S}}], b\in\{0,1\}})$ to $\mathcal{A}$.
4. **Experiment Outcome:** $\mathcal{A}$ outputs a bit b' which is the output of the experiment.

Lemma 4.11. *For all adversaries $\mathcal{A}$,*

$$\left| \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{6,\mathsf{Bound}_{\mathcal{A}},1}(1^\lambda)] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_7(1^\lambda)] \right| = 0.$$

where $\mathsf{Bound}_{\mathcal{A}}$ is a bound on the runtime of $\mathcal{A}$.

Proof. The hybrids are identical. □

Thus, our lemmas give us the following corollary:

Corollary 4.12. *If*

- SKE *is a secure secret-key encryption scheme,*
- $\mathsf{PRF}_1, \mathsf{PRF}_2, \mathsf{PRF}_3, \mathsf{PRF}_p$ *are secure pseudorandom function families,*
- GC *is a secure garbling scheme,*

then $\mathsf{One\text{-}sFE}$ *is a single-key, single-ciphertext, function-selective-SIM-secure, secret-key sFE scheme for* $\mathsf{P/Poly}$*.*

Proof. The corollary immediately follows from Lemmas 4.3-4.11. $\square$

Corollary 4.12 then implies Theorem 4.1 since we can instantiate each of the required primitives from OWFs.

5 Bootstrapping to a Q-Bounded Public-Key sFE Scheme

In this section, we prove the following theorem:

Theorem 5.1. *Assuming*

1. *a Q-bounded, adaptive-IND-secure, public-key (resp. secret-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 Q-bounded, semi-adaptive-function-selective-IND-secure, public-key (resp. secret-key) sFE scheme for $\mathsf{P/Poly}$.

Then by applying Theorem 4.1 and a theorem from [AV19], we get our main theorem.[19]

Theorem 5.2. *Assuming the existence of a public-key (resp. secret-key) encryption scheme, there exists a Q-bounded, semi-adaptive-function-selective-IND-secure, public-key (resp. secret-key)* sFE *scheme for* $\mathsf{P/Poly}$ *for any polynomial $Q = Q(\lambda)$ of the security parameter λ.*

To prove Theorem 5.1, we build an sFE scheme from the following tools.

Tools.

- $\mathsf{One\text{-}sFE} = (\mathsf{One\text{-}sFE.Setup}, \mathsf{One\text{-}sFE.EncSetup}, \mathsf{One\text{-}sFE.Enc}, \mathsf{One\text{-}sFE.KeyGen}, \mathsf{One\text{-}sFE.Dec})$: A single-key, single-ciphertext, function-selective-IND-secure, secret-key sFE scheme for $\mathsf{P/Poly}$.
- The following primitives can be built from a Q-bounded, adaptive-IND-secure, public-key (resp. secret-key) FE 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.
  - $\mathsf{SKE} = (\mathsf{SKE.Setup}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$: A secure secret-key encryption scheme with pseudorandom ciphertexts.
  - $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.Enc}, \mathsf{FE.KeyGen}, \mathsf{FE.Dec})$: A Q-bounded, selective-IND-secure, public-key (resp. secret-key) FE scheme for $\mathsf{P/Poly}$.
  - $\mathsf{FPFE} = (\mathsf{FPFE.Setup}, \mathsf{FPFE.Enc}, \mathsf{FPFE.KeyGen}, \mathsf{FPFE.Dec})$: A Q-bounded, function-private, function-selective-IND-secure, secret-key FE scheme for $\mathsf{P/Poly}$.

Instantiation of the Tools. Let AdFE be a Q-bounded, adaptive-IND-secure, public-key (resp. secret-key) FE scheme for $\mathsf{P/Poly}$.

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

[19] In particular, [AV19] show how to build a Q-bounded, adaptive-IND-secure, public-key (resp. secret-key) FE scheme for $\mathsf{P/Poly}$ from a public-key (resp. secret-key) encryption scheme. We show in Theorem 4.1 how to build a single-key, single-ciphertext, function-selective-SIM-secure, secret-key sFE scheme for $\mathsf{P/Poly}$ from OWFs. Note that OWFs can be built using secret-key (or public-key) encryption and SIM security can be easily shown to imply the equivalent IND security.

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:

	Security Parameter	Input Size	Output Size	Function Size	State Size
One-sFE	λ	$\ell_{\mathcal{X}}$	$\ell_{\mathcal{Y}}$	$\ell_{\mathcal{F}}$	$\ell_{\mathcal{S}}$
PRF	λ	λ	5λ		
PRF2	λ	λ	λ		
SKE	λ				
FPFE	λ	$\ell_{\mathsf{FPFE}.m}$	$\ell_{\mathsf{FPFE}.out}$	ℓ_H	
FE	λ	$\ell_{\mathsf{FE}.m}$	$\ell_{\mathsf{FE}.out}$	ℓ_G	

where we define

- $\ell_{\mathsf{FPFE}.m} = 2\lambda + 2\ell_{\mathcal{X}} + \ell_{\mathsf{One\text{-}sFE.ct}}$ where $\ell_{\mathsf{One\text{-}sFE.ct}}$ is the size of ciphertexts of One-sFE.
- $\ell_{\mathsf{FPFE}.out} = \ell_{\mathsf{One\text{-}sFE.ct}}$ where $\ell_{\mathsf{One\text{-}sFE.ct}}$ is the size of ciphertexts of One-sFE.
- ℓ_H is the maximum of
  - the size of $H[\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, \mathsf{PRF2}.K]$ defined in Figure 10
  - the size of $H'[\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, \mathsf{PRF2}.K]$ defined in Figure 11
  - the size of H^* defined in Figure 12

  for any master secret key One-sFE.msk and encryption state One-sFE.Enc.st of One-sFE, and any key $\mathsf{PRF2}.K$ of PRF2.
- $\ell_{\mathsf{FE}.m} = \ell_{\mathsf{FPFE.msk}} + \ell_{\mathsf{PRF}.K} + 1 + \ell_{\mathsf{SKE.sk}}$ where $\ell_{\mathsf{FPFE.msk}}$ is the size of master secret keys of FPFE, $\ell_{\mathsf{PRF}.K}$ is the size of keys of PRF, and $\ell_{\mathsf{SKE.sk}}$ is the size of keys of SKE.
- $\ell_{\mathsf{FE}.out} = \ell_{\mathsf{One\text{-}sFE.sk}} + \ell_{\mathsf{FPFE}.sk}$ where $\ell_{\mathsf{One\text{-}sFE.sk}}$ is the size of secret keys of One-sFE and $\ell_{\mathsf{FPFE.sk}}$ is the size of function keys of FPFE.
- ℓ_G is the maximum size of $G_{f,s,c}$ defined in Figure 9 for any $f \in \mathcal{F}[\ell_{\mathcal{F}}, \ell_{\mathcal{S}}, \ell_{\mathcal{X}}, \ell_{\mathcal{Y}}]$, $s \in \{0,1\}^\lambda$, and c of size $\ell_{\mathsf{SKE.ct}}$ where $\ell_{\mathsf{SKE.ct}}$ is the size of ciphertexts of SKE when encrypting plaintexts of size $\ell_{\mathsf{SKE}.m} = \ell_{\mathsf{FE}.out}$.

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

Remark 5.3. 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 pseudorandom generator (PRG) of appropriate stretch and then runs the original algorithm. Note that this replacement can be implemented with OWFs and 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).

5.2 Construction

We now construct our Q-bounded streaming functional encryption scheme sFE.

Remark 5.4. We provide our construction in both the secret-key and public-key settings. In the secret-key seting, sFE and FE are both secret-key schemes, and in the public-key setting, sFE and FE are both public-key schemes. We use input $\mathsf{MSK}/\mathsf{MPK}$ to denote that the algorithm receives MSK in the secret-key setting and MPK in the public-key setting.

Remark 5.5. Recall that for notational convenience, we may omit the security, input size, output size, function size, or state size parameters from our algorithms. For information on these parameters, please see the parameter section above.

- $\mathsf{sFE.Setup}(1^\lambda, 1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{S}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}})$:
  1. *Secret-Key Setting:*
     (a) $\mathsf{FE.msk} \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
     (b) Output $\mathsf{MSK} = \mathsf{FE.msk}$.
  2. *Public-Key Setting:*
     (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
     (b) Output $(\mathsf{MPK}, \mathsf{MSK}) = (\mathsf{FE.mpk}, \mathsf{FE.msk})$.
- $\mathsf{sFE.EncSetup}(\mathsf{MSK}/\mathsf{MPK})$:
  1. *Secret-Key Setting:* Parse $\mathsf{MSK} = \mathsf{FE.msk}$. Set $\mathsf{FE.ek} = \mathsf{FE.msk}$.
  2. *Public-Key Setting:* Parse $\mathsf{MPK} = \mathsf{FE.mpk}$. Set $\mathsf{FE.ek} = \mathsf{FE.mpk}$.
  3. $\mathsf{PRF}.K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
  4. $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$.
  5. $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.ek}, (\mathsf{FPFE.msk}, \mathsf{PRF}.K, 0, 0^{\ell_{\mathsf{SKE.sk}}}))$.
  6. Output $\mathsf{Enc.ST} = (\mathsf{FPFE.msk}, \mathsf{FE.ct})$.
- $\mathsf{sFE.Enc}(\mathsf{MSK}/\mathsf{MPK}, \mathsf{Enc.ST}, i, x_i)$:
  1. Parse $\mathsf{Enc.ST} = (\mathsf{FPFE.msk}, \mathsf{FE.ct})$.
  2. $t_i \leftarrow \{0,1\}^\lambda$.
  3. $\mathsf{FPFE.ct}_i \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (i, t_i, x_i, 0^{\ell_\mathcal{X}}, 0^{\ell_{\mathsf{One\text{-}sFE.ct}}}))$.
  4. If $i = 1$, output $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.ct}_1)$.
  5. Else, output $\mathsf{CT}_i = \mathsf{FPFE.ct}_i$.
- $\mathsf{sFE.KeyGen}(\mathsf{MSK}, f)$:
  1. Parse $\mathsf{MSK} = \mathsf{FE.msk}$.
  2. $s \leftarrow \{0,1\}^\lambda$.
  3. $c \leftarrow \{0,1\}^{\ell_{\mathsf{SKE.ct}}}$.
  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 $\mathsf{SK}_f = \mathsf{FE.sk}_G$.

$G[f,s,c](\mathsf{FPFE.msk}, \mathsf{PRF}.K, \alpha, \mathsf{SKE.sk})$:

1. If $\alpha = 0$,
   (a) $(r_{\mathsf{Setup}}, r_{\mathsf{EncSetup}}, r_{\mathsf{KeyGen}}, r_{\mathsf{PRF2}}, r_H) \leftarrow \mathsf{PRF.Eval}(\mathsf{PRF}.K, s)$.
   (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) Let $H = H[\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, \mathsf{PRF2}.K]$ as defined in Figure 10.
   (g) $\mathsf{FPFE.sk}_H \leftarrow \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H; r_H)$.
   (h) Output $(\mathsf{One\text{-}sFE.sk}_f, \mathsf{FPFE.sk}_H)$.
2. Else,
   (a) Output $(\mathsf{One\text{-}sFE.sk}_f, \mathsf{FPFE.sk}_H) \leftarrow \mathsf{SKE.Dec}(\mathsf{SKE.sk}, c)$.

Figure 9: Definition of $G[f,s,c]$

$H[\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, \mathsf{PRF2}.K](i, t_i, x_i, x'_i, v_i)$:

1. $r_i \leftarrow \mathsf{PRF2.Eval}(\mathsf{PRF2}.K, t_i)$.
2. Output $\mathsf{One\text{-}sFE.ct}_i \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, i, x_i; r_i)$.

Figure 10: Definition of $H[\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, \mathsf{PRF2}.K]$

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

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

$H'[\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, \mathsf{PRF2}.K](i, t_i, x_i, x_i', v_i)$:

1. $r_i \leftarrow \mathsf{PRF2.Eval}(\mathsf{PRF2}.K, t_i)$.
2. Output $\mathsf{One\text{-}sFE.ct}_i \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, i, x_i'; r_i)$.

Figure 11: Definition of $H'[\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, \mathsf{PRF2}.K]$

$H^*(i, t_i, x_i, x_i', v_i)$: Output v_i.

Figure 12: Definition of H^*

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 our FE scheme on security parameter 1^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 $(\mathsf{FE.ct}, \{\mathsf{FPFE.ct}_i\}_{i\in[n]})$, 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 function key $\mathsf{FPFE.sk}_H$ for H which has $\mathsf{One\text{-}sFE.msk}$ hardwired into it. Then, for $i \in [n]$, we can combine $\mathsf{FPFE.ct}_i$ and $\mathsf{FPFE.sk}_H$ 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 $\mathsf{SK}_f = \mathsf{FE.sk}_G$, $\mathsf{CT}_1 = (\mathsf{FE.ct}, \mathsf{FPFE.ct}_1)$, and $\mathsf{CT}_i = \mathsf{FPFE.ct}_i$ for $i \in [n]\backslash\{1\}$. Then, by correctness of FE, except with negligible probability,

$$\begin{aligned}\mathsf{FE.Dec}(\mathsf{FE.sk}_G, \mathsf{FE.ct}) &= G[f, s, c](\mathsf{FPFE.msk}, \mathsf{PRF}.K, 0, 0^{\ell_{\mathsf{SKE.sk}}}) \\ &= (\mathsf{One\text{-}sFE.sk}_f, \mathsf{FPFE.sk}_H)\end{aligned}$$

where $\mathsf{One\text{-}sFE.sk}_f$ is a One-sFE function key for f generated under $\mathsf{One\text{-}sFE.msk}$, and $\mathsf{FPFE.sk}_H$ is an FPFE function key for $H[\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, \mathsf{PRF2}.K]$ as defined by

$(r_{\mathsf{Setup}}, r_{\mathsf{EncSetup}}, r_{\mathsf{KeyGen}}, r_{\mathsf{PRF2}}, r_H) \leftarrow \mathsf{PRF.Eval}(\mathsf{PRF}.K, s)$

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

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

$\mathsf{One\text{-}sFE.sk}_f \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}, f; r_{\mathsf{KeyGen}})$

$\mathsf{PRF2}.K \leftarrow \mathsf{PRF2.Setup}(1^\lambda; r_{\mathsf{PRF2}})$

Let $H = H[\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, \mathsf{PRF2}.K]$ as defined in Figure 10.

$\mathsf{FPFE.sk}_H \leftarrow \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H; r_H)$.

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

$$\begin{aligned}\mathsf{FPFE.Dec}(\mathsf{FPFE.sk}_H, \mathsf{FPFE.ct}_i) &= H[\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, \mathsf{PRF2}.K](i, t_i, x_i, 0^{\ell_{\mathcal{X}}}, 0^{\ell_{\mathsf{One\text{-}sFE.ct}}}) \\ &= \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}, \mathsf{One\text{-}sFE.Enc.st}, i, x_i; \mathsf{PRF2.Eval}(\mathsf{PRF2}.K, t_i)) \\ &= \mathsf{One\text{-}sFE.ct}_i\end{aligned}$$

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, $y = y_1 \dots y_n = f(x)$. Therefore, for $i = 1$ and using the values we defined above,

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

For $i > 1$, using the values defined above,

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

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

5.4 Security

In this section, we prove that sFE is Q-bounded, semi-adaptive-function-selective-IND-secure.

5.4.1 Proof Overview

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

- $\mathbf{Hybrid}_0^{\mathcal{A}}$: This is the real world experiment with $b = 0$.
- For each stream id w, we proceed through a sequence of hybrids to swap the encryption of stream $x_w^{(0)} = x_{w,1}^{(0)} x_{w,2}^{(0)} \dots x_{w,n}^{(0)}$ with an encryption of stream $x_w^{(1)} = x_{w,1}^{(1)} x_{w,2}^{(1)} \dots x_{w,n}^{(1)}$.
  - $\mathbf{Hybrid}_{1,w,0}^{\mathcal{A}}$: For stream identities $\mathsf{id} < w$, we encrypt $x_{\mathsf{id}}^{(1)}$ instead of $x_{\mathsf{id}}^{(0)}$. For $w = 1$, this hybrid is identical to $\mathbf{Hybrid}_0^{\mathcal{A}}$.
  - $\mathbf{Hybrid}_{1,w,1}^{\mathcal{A}}$: For each function key $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$, we hardwire the output of G_j on stream w into the ciphertext c_j embedded within G_j. Indistinguishability follows from the pseudorandom ciphertext property of SKE.
  - $\mathbf{Hybrid}_{1,w,2}^{\mathcal{A}}$: We swap ciphertext $\mathsf{FE.ct}_w$ from an encryption of $(\mathsf{FPFE.msk}_w, \mathsf{PRF}.K_w, 0, 0^{\ell_{\mathsf{SKE.sk}}})$ to an encryption of $(0^{\ell_{\mathsf{FPFE.msk}}}, 0^{\ell_{\mathsf{PRF}.K}}, 1, \mathsf{SKE.sk})$. Now, when decrypting $\mathsf{FE.ct}_w$ with $\mathsf{FE.sk}_{G_j}$, we will invoke the $\alpha = 1$ branch of G_j, and output the value encrypted within c_j. Since the value within c_j had been previously set to the correct output value, indistinguishability follows from the security of FE. This change also removes $(\mathsf{FPFE.msk}_w, \mathsf{PRF}.K_w)$ from the hybrid..
  - $\mathbf{Hybrid}_{1,w,3}^{\mathcal{A}}$: We replace the values generated by $\mathsf{PRF}.K_w$ with true randomness. This includes the randomness used to generate the $\mathsf{One\text{-}sFE}$, $\mathsf{PRF2}$, and FPFE keys for stream w. Indistinguishability follows from the security of PRF.
  - Within each ciphertext $\mathsf{FPFE.ct}_i$ for stream w, we encrypt both $x_{w,i}^{(0)}$ and $x_{w,i}^{(1)}$. Then, for each $k \in [Q]$, we proceed through a sequence of hybrids to change $\mathsf{FPFE.sk}_{H_{k,w}}$ from a function key which decrypts using $x_{w,i}^{(0)}$ to a function key which decrypts using $x_{w,i}^{(1)}$.
    * $\mathbf{Hybrid}_{1,w,4,k,0}^{\mathcal{A}}$: Within each ciphertext $\mathsf{FPFE.ct}_{w,i}$ for stream w, we additionally encrypt $x_{w,i}^{(1)}$ and $v_{k,w,i} = \mathsf{FPFE.Dec}(\mathsf{FPFE.sk}_{H_{k,w}}, \mathsf{FPFE.ct}_{w,i})$. For $j < k$, we set function $H_{j,w}$ to a function which computes its output using $x_w^{(1)}$. For $j = k$, we set $H_{j,w}$ to a function that simply outputs $v_{k,w,i}$. For $k = 1$, this hybrid is indistinguishable from $\mathbf{Hybrid}_{1,w,3}^{\mathcal{A}}$ by the security of FPFE.
    * $\mathbf{Hybrid}_{1,w,4,k,1}^{\mathcal{A}}$: We exchange the randomness generated by $\mathsf{PRF2}.K_w$ with true randomness. This randomness is used to compute $v_{k,w,i}$. Indistinguishability follows from the security of $\mathsf{PRF2}$.
    * $\mathbf{Hybrid}_{1,w,4,k,2}^{\mathcal{A}}$: We invoke the security of $\mathsf{One\text{-}sFE}$ to change each $v_{k,w,i}$ from an encryption of $x_{w,i}^{(0)}$ to an encryption of $x_{w,i}^{(1)}$. Indistinguishability follows from the security of $\mathsf{One\text{-}sFE}$.
    * $\mathbf{Hybrid}_{1,w,4,k,3}^{\mathcal{A}}$: We revert back to using $\mathsf{PRF2}.K_w$ to compute the randomness needed for determining $v_{k,w,i}$. Indistinguishability follows from the security of $\mathsf{PRF2}$.
    * $\mathbf{Hybrid}_{1,w,4,k,4}^{\mathcal{A}}$: We change $H_{k,w}$ from a function that simply outputs $v_{k,w,i}$ to a function that decrypts using $x_w^{(1)}$. Indistinguishability follows from the security of

FPFE. Additionally, the indistinguishability of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,4}$ and $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k+1,0}$ follows by the security of FPFE.

- $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,5}$: This is identical to $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,Q,4}$ Observe that every $\mathsf{FPFE.sk}_{H_{k,w}}$ now decrypts using $x^{(1)}_{w,i}$ instead of $x^{(0)}_{w,i}$.
- $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,6}$: Within each ciphertext $\mathsf{FPFE.ct}_{w,i}$ for stream identity w, we encrypt $x^{(1)}_{w,i}$, but no longer encrypt either $x^{(0)}_{w,i}$ or $v_{k,w,i}$. We also change each $\mathsf{FPFE.sk}_{H_{k,w}}$ back to its original value. Since we now only encrypt $x^{(1)}_{w}$, we will continue to compute using stream $x^{(1)}_{w}$. Indistinguishability follows from the security of FPFE.
- $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,7}$: We revert back to using $\mathsf{PRF}.K_w$ to generate the randomness needed for computing the One-sFE, PRF2, FPFE keys for stream w. Indistinguishability follows from the security of PRF.
- $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,8}$: We change $\mathsf{FE.ct}_w$ back to its original value. Indistinguishability follows from the security of FE.
- $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,9}$: We change the ciphertexts c_j back to pseudorandom values. Indistinguishability follows from the pseudorandom ciphertext property of SKE. This hybrid is also identical to $\mathbf{Hybrid}^{\mathcal{A}}_{1,w+1,0}$.

- $\mathbf{Hybrid}^{\mathcal{A}}_{2}$: This is the real world experiment with $b = 1$. This hybrid is identical to $\mathbf{Hybrid}^{\mathcal{A}}_{1,\mathsf{Bound}_{\mathcal{A}},9}$ where $\mathsf{Bound}_{\mathcal{A}}$ is a bound on the runtime of $\mathcal{A}$ and thus an implicit bound on the number of stream identities queries by $\mathcal{A}$.

5.4.2 Formal Proof

We now formally prove Theorem 5.1 via a hybrid argument. We will be simultaneously proving both the secret-key and public-key versions of the theorem. The differences are explicitly highlighted in the hybrids and theorem statements.

Remark 5.6 (Multiple Pairs of Challenge Streams)**.** In the secret-key setting (Definition A.9), the adversary is allowed to make message queries across multiple pairs of challenge streams. For each message query, the adversary may choose to either append new stream values to an existing pair of challenge streams or to start a new pair of challenge streams from index 1. The adversary indicates which pair of streams each message query belongs to using a stream identity id. We assume without loss of generality that whenever the adversary makes a query to a new stream identity id, the adversary sets $\mathsf{id} = i$ where i is the number of unique stream identities queried thus far including this one. Note that this can be accomplished via relabeling.

In the public-key setting (Definition 3.14), the adversary is only allowed to make message queries for one pair of challenge streams. Thus, the stream identity is not specified in the security game since we do not need to differentiate between multiple pairs of challenge streams. However, since we are simultaneously proving security in both the public-key and secret-key settings, we will nevertheless refer to the stream identity in the following hybrids and proof of security. In the public-key setting, we will assume that any message query made by the adversary has a default stream identity $\mathsf{id} = 1$.[20]

Remark 5.7 (Abort Condition)**.** We require all of our hybrids to immediately halt and output 0 if the adversary ever aborts or if it at any point some function query f submitted by the adversary yields different outputs on any of the challenge message streams submitted so far (i.e. if $f(x^{(0)}_{\mathsf{id}}) \neq f(x^{(1)}_{\mathsf{id}})$ for some function query f submitted by the adversary where $\{(x^{(0)}_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i})\}_{i\in[t]}$ are the message queries submitted so far under some stream identity id, $x^{(0)}_{\mathsf{id}} = x^{(0)}_{\mathsf{id},1} \ldots x^{(0)}_{\mathsf{id},t}$, and $x^{(1)}_{\mathsf{id}} = x^{(1)}_{\mathsf{id},1} \ldots x^{(1)}_{\mathsf{id},t}$). For notational simplicity, we will omit this requirement from the description of our hybrids.

Remark 5.8 (Bootstrapping in the Adaptive Setting)**.** In fact, if our $\mathsf{One\text{-}sFE}$ scheme was adaptively secure, then our bootstrapping would produce an adaptively secure sFE scheme.

More precisely, assuming (1) a Q-bounded, adaptive-IND-secure, public-key (resp. secret-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 a Q-bounded *adaptive*-IND-secure, public-key (resp. secret-key) sFE scheme for P/Poly.

The construction is identical to the construction given in section 5.2. The proof of security is also identical except that (1) we change step 5 in the hybrids to "The adversary may make up to Q function queries and any polynomial number of message queries *in any order*", and (2) we use the adaptive security of $\mathsf{One\text{-}sFE}$ to prove Lemma 5.15. It is easy to check that this modified proof works regardless of the order in which message and function queries are received.

[20] Alternatively, we could allow the adversary in the public-key setting to also make message queries across multiple pairs of challenge streams in a similar manner as in the secret-key setting. By a standard hybrid argument, this modified security definition is equivalent to Definition 3.14.

$\mathbf{Hybrid}_0^{\mathcal{A}}(1^\lambda)$: This is the real world experiment with $b = 0$. Note that we have rearranged some steps and thus will compute the **Encryption Setup** step early on rather than when we receive the message queries. This does not affect the outcome of the experiment since we can receive at most $\mathsf{Bound}_{\mathcal{A}}$ stream identities id during the hybrid where $\mathsf{Bound}_{\mathcal{A}}$ is a bound on the runtime of $\mathcal{A}$.

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:**
   - *Secret-Key Setting:*
     (a) $\mathsf{FE.msk} \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
     (b) $\mathsf{FE.ek} = \mathsf{FE.msk}$.
   - *Public-Key Setting:*
     (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
     (b) $\mathsf{FE.ek} = \mathsf{FE.mpk}$.
     (c) Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.

3. **Encryption Setup:** For $\mathsf{id} \in [\mathsf{Bound}_{\mathcal{A}}]$ where $\mathsf{Bound}_{\mathcal{A}}$ is a bound on the runtime of $\mathcal{A}$.
   (a) $\mathsf{PRF}.K_{\mathsf{id}} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
   (b) $\mathsf{FPFE.msk}_{\mathsf{id}} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$.
   (c) $\mathsf{FE.ct}_{\mathsf{id}} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.ek}, (\mathsf{FPFE.msk}_{\mathsf{id}}, \mathsf{PRF}.K_{\mathsf{id}}, 0, 0^{\ell_{\mathsf{SKE.sk}}}))$.

4. **Precompute Values:** Do nothing. (Will be added in a later hybrid.)

5. The adversary can make up to Q function queries followed by any polynomial number of message queries.
   (a) **Function Query:** 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:
      i. $s_j \leftarrow \{0,1\}^\lambda$.
      ii. $c_j \leftarrow \{0,1\}^{\ell_{\mathsf{SKE.ct}}}$.
      iii. Let $G_j = G[f_j, s_j, c_j]$ as defined in Figure 9 (page 50).
      iv. $\mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_j)$.
      v. Send $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to the adversary.
   (b) **Message Query:** For the i^{th} message query made to stream identity id, $\mathcal{A}$ outputs a message pair $(x^{(0)}_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i})$ where $x^{(0)}_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i} \in \{0,1\}^{\ell_{\mathcal{X}}}$.
      i. $t_{\mathsf{id},i} \leftarrow \{0,1\}^\lambda$.
      ii. $\mathsf{FPFE.ct}_{\mathsf{id},i} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}_{\mathsf{id}}, (i, t_{\mathsf{id},i}, x^{(0)}_{\mathsf{id},i}, 0^{\ell_{\mathcal{X}}}, 0^{\ell_{\mathsf{One\text{-}sFE.ct}}}))$.
      iii. If $i = 1$, set $\mathsf{CT}_{\mathsf{id},1} = (\mathsf{FE.ct}_{\mathsf{id}}, \mathsf{FPFE.ct}_{\mathsf{id},1})$. Else, set $\mathsf{CT}_{\mathsf{id},i} = \mathsf{FPFE.ct}_{\mathsf{id},i}$.
      iv. Send $\mathsf{CT}_{\mathsf{id},i}$ to the adversary.

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

$\mathbf{Hybrid}^{\mathcal{A}}_{1,w,0}(1^\lambda)$: For stream identities $\mathsf{id} < w$, we encrypt $x^{(1)}_{\mathsf{id}}$ instead of $x^{(0)}_{\mathsf{id}}$.

This is the same as $\mathbf{Hybrid}^{\mathcal{A}}_0$ except that we change the following steps:

5b. **Message Query:** For the i^{th} message query made to stream identity id, $\mathcal{A}$ outputs a message pair $(x^{(0)}_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i})$ where $x^{(0)}_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i} \in \{0,1\}^{\ell_{\mathcal{X}}}$.

(a) $t_{\mathsf{id},i} \leftarrow \{0,1\}^\lambda$.

(b) If $\mathsf{id} < w$, $\mathsf{FPFE.ct}_{\mathsf{id},i} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}_{\mathsf{id}}, (i, t_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i}, 0^{\ell_{\mathcal{X}}}, 0^{\ell_{\mathsf{One\text{-}sFE.ct}}}))$.

(c) If $\mathsf{id} \geq w$, $\mathsf{FPFE.ct}_{\mathsf{id},i} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}_{\mathsf{id}}, (i, t_{\mathsf{id},i}, x^{(0)}_{\mathsf{id},i}, 0^{\ell_{\mathcal{X}}}, 0^{\ell_{\mathsf{One\text{-}sFE.ct}}}))$.

(d) If $i = 1$, set $\mathsf{CT}_{\mathsf{id},1} = (\mathsf{FE.ct}_{\mathsf{id}}, \mathsf{FPFE.ct}_{\mathsf{id},1})$. Else, set $\mathsf{CT}_{\mathsf{id},i} = \mathsf{FPFE.ct}_{\mathsf{id},i}$.

(e) Send $\mathsf{CT}_{\mathsf{id},i}$ to the adversary.

Lemma 5.9. *For all adversaries $\mathcal{A}$,*

$$\left| \Pr[\mathbf{Hybrid}^{\mathcal{A}}_0(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,1,0}(1^\lambda) = 1] \right| = 0$$

Proof. The hybrids are identical since we have assumed that each stream identity $\mathsf{id} \geq 1$. (See Remark 5.6.) $\square$

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

$$(\mathsf{One\text{-}sFE.sk}_{f_j,w}, \mathsf{FPFE.sk}_{H_{j,w}}) = G_j(\mathsf{FPFE.msk}_w, \mathsf{PRF}.K_w, 0, 0^{\ell_{\mathsf{SKE.sk}}})$$

which would be generated in the real world experiment. This will allow us to later switch to the $\alpha = 1$ branch in G_j using the security of FE. Observe that the values being hardcoded into c_j can be computed before receiving any message queries.

This is the same as $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,0}$ except that we change the following steps:

2. **Setup:**
   - *Secret-Key Setting:*
     (a) $\mathsf{SKE.sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
     (b) $\mathsf{FE.msk} \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
     (c) $\mathsf{FE.ek} = \mathsf{FE.msk}$.
   - *Public-Key Setting:*
     (a) $\mathsf{SKE.sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
     (b) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
     (c) $\mathsf{FE.ek} = \mathsf{FE.mpk}$.
     (d) Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.

4. **Precompute Values:** For $j \in [Q]$,
   (a) $s_j \leftarrow \{0,1\}^\lambda$.
   (b) $(r_{\mathsf{Setup},j,w}, r_{\mathsf{EncSetup},j,w}, r_{\mathsf{KeyGen},j,w}, r_{\mathsf{PRF2},j,w}, r_{H,j,w}) \leftarrow \mathsf{PRF.Eval}(\mathsf{PRF}.K_w, s_j)$.
   (c) $\mathsf{One\text{-}sFE.msk}_{j,w} \leftarrow \mathsf{One\text{-}sFE.Setup}(1^\lambda; r_{\mathsf{Setup},j,w})$.
   (d) $\mathsf{One\text{-}sFE.Enc.st}_{j,w} \leftarrow \mathsf{One\text{-}sFE.EncSetup}(\mathsf{One\text{-}sFE.msk}_{j,w}; r_{\mathsf{EncSetup},j,w})$.
   (e) $\mathsf{PRF2}.K_{j,w} \leftarrow \mathsf{PRF2.Setup}(1^\lambda; r_{\mathsf{PRF2},j,w})$.
   (f) Let $H_{j,w} = H[\mathsf{One\text{-}sFE.msk}_{j,w}, \mathsf{One\text{-}sFE.Enc.st}_{j,w}, \mathsf{PRF2}.K_{j,w}]$ as defined in Figure 10 (page 50).
   (g) $\mathsf{FPFE.sk}_{H_{j,w}} \leftarrow \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}_w, H_{j,w}; r_{H,j,w})$.

5a. **Function Query:** 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) $\cancel{s_j \leftarrow \{0,1\}^\lambda.}$
   (b) $\mathsf{One\text{-}sFE.sk}_{f_j,w} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_{j,w}, f_j; r_{\mathsf{KeyGen},j,w})$.
   (c) $c_{j,w} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, (\mathsf{One\text{-}sFE.sk}_{f_j,w}, \mathsf{FPFE.sk}_{H_{j,w}}))$.
   (d) Let $G_j = G[f_j, s_j, c_{j,w}]$ as defined in Figure 9 (page 50).
   (e) $\mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_j)$.
   (f) Send $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to the adversary.

Lemma 5.10. *If* SKE *has pseudorandom ciphertexts, then for all PPT adversaries* $\mathcal{A}$ *and all* $w \in [\mathsf{Bound}_\mathcal{A}]$*,*

$$\left| \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,0}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,1}(1^\lambda) = 1] \right| \leq \mathsf{negl}(\lambda)$$

Proof. Suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$ and a $w \in [\mathsf{Bound}_{\mathcal{A}}]$ such that

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,0}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,1}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{6}$$

We build a PPT adversary $\mathcal{B}$ that breaks the pseudorandom ciphertext property of SKE. $\mathcal{B}$ first runs steps 1-3 as in $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,1}$ except that $\mathcal{B}$ does not compute $\mathsf{SKE.sk}$. For $j \in [Q]$, $\mathcal{B}$ computes $(s_j, \mathsf{One\text{-}sFE.msk}_{j,w}, r_{\mathsf{KeyGen},j,w}, \mathsf{FPFE.sk}_{H_{j,w}})$ as in step 4 of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,1}$.

- For each function query f_j output by $\mathcal{A}$, $\mathcal{B}$ does the following: $\mathcal{B}$ computes $\mathsf{One\text{-}sFE.sk}_{f_j,w} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_{j,w}, f_j; r_{\mathsf{KeyGen},j,w})$ and sends $m_{j,w} = (\mathsf{One\text{-}sFE.sk}_{f_j,w}, \mathsf{FPFE.sk}_{H_{j,w}})$ to its SKE challenger. $\mathcal{B}$ receives $c_{j,w}$ which is either an encryption of $m_{j,w}$ or a uniform random value. $\mathcal{B}$ computes $\mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_j)$ where $G_j = G[f_j, s_j, c_{j,w}]$, and sends $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to the adversary.

- For each message query $(x^{(0)}_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i})$ output by $\mathcal{A}$, $\mathcal{B}$ computes $\mathsf{CT}_{\mathsf{id},i}$ as in step 5b of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,1}$, and sends $\mathsf{CT}_{\mathsf{id},i}$ to $\mathcal{A}$.

After $\mathcal{A}$ is done making queries, $\mathcal{A}$ outputs b' which $\mathcal{B}$ also outputs. If the experiment for $\mathcal{A}$ aborts for any reason, $\mathcal{B}$ instead outputs 0. Observe that if every $c_{j,w}$ is an independent uniform random value, then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,0}$, and if each $c_{j,w}$ is an encryption of $m_{j,w}$, then $\mathcal{B}$ emulates $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,1}$. Additionally, $\mathcal{B}$ does not need to know $\mathsf{SKE.sk}$ to carry out this experiment. Thus, by Equation 6, this means that $\mathcal{B}$ breaks the pseudorandom ciphertext property of SKE as $\mathcal{B}$ can distinguish between receiving random values and valid ciphertexts with non-negligible probability. $\square$

$\mathbf{Hybrid}^{\mathcal{A}}_{1,w,2}(1^\lambda)$: We change the message encrypted in $\mathsf{FE.ct}_w$ so that we use the $\alpha = 1$ branch of every G_j. This allows us to remove $\mathsf{FPFE.msk}_w$ and $\mathsf{PRF}.K_w$ from $\mathsf{FE.ct}_w$.

This is the same as $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,1}$ except that we change the following steps:

3. **Encryption Setup:** For $\mathsf{id} \in [\mathsf{Bound}_{\mathcal{A}}]$ where $\mathsf{Bound}_{\mathcal{A}}$ is a bound on the runtime of $\mathcal{A}$.
   (a) $\mathsf{PRF}.K_{\mathsf{id}} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
   (b) $\mathsf{FPFE.msk}_{\mathsf{id}} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$.
   (c) If $\mathsf{id} \neq w$, $\mathsf{FE.ct}_{\mathsf{id}} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.ek}, (\mathsf{FPFE.msk}_{\mathsf{id}}, \mathsf{PRF}.K_{\mathsf{id}}, 0, 0^{\ell_{\mathsf{SKE.sk}}}))$.
   (d) If $\mathsf{id} = w$, $\mathsf{FE.ct}_{\mathsf{id}} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.ek}, (0^{\ell_{\mathsf{FPFE.msk}}}, 0^{\ell_{\mathsf{PRF}.K}}, 1, \mathsf{SKE.sk}))$.

Lemma 5.11. *If* FE *is a public-key (resp. secret-key) Q-bounded, selective-IND-secure scheme, then for all PPT adversaries $\mathcal{A}$ and all $w \in [\mathsf{Bound}_{\mathcal{A}}]$, for the public-key (resp. secret-key) version of the hybrids,*

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

Proof. Suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$ and a $w \in [\mathsf{Bound}_{\mathcal{A}}]$ such that

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

We build a PPT adversary $\mathcal{B}$ that breaks the Q-bounded, selective-IND-security of FE. $\mathcal{B}$ first runs step 1 of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,2}$ and computes $(\mathsf{SKE.sk}, \{(\mathsf{PRF}.K_{\mathsf{id}}, \mathsf{FPFE.msk}_{\mathsf{id}})\}_{\mathsf{id} \in [\mathsf{Bound}_{\mathcal{A}}]})$ as in steps 2-3 of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,2}$.

$\mathcal{B}$ sends challenge message pair $(m_{0,w}, m_{1,w}) = ((\mathsf{FPFE.msk}_w, \mathsf{PRF}.K_w, 0, 0^{\ell_{\mathsf{SKE.sk}}}), (0^{\ell_{\mathsf{FPFE.msk}}}, 0^{\ell_{\mathsf{PRF}.K}}, 1, \mathsf{SKE.sk}))$ to its FE challenger and receives $\mathsf{FE.ct}_w$ which is an encryption of either $m_{0,w}$ or $m_{1,w}$.

For $\mathsf{id} \in [\mathsf{Bound}_{\mathcal{A}}] \backslash \{w\}$, $\mathcal{B}$ sets $m_{\mathsf{id}} = (\mathsf{FPFE.msk}_{\mathsf{id}}, \mathsf{PRF}.K_{\mathsf{id}}, 0, 0^{\ell_{\mathsf{SKE.sk}}})$. In the secret-key setting, for $\mathsf{id} \in [\mathsf{Bound}_{\mathcal{A}}] \backslash \{w\}$, $\mathcal{B}$ sends challenge message pair $(m_{\mathsf{id}}, m_{\mathsf{id}})$ to its FE challenger and receives an encryption $\mathsf{FE.ct}_{\mathsf{id}}$ of m_{id}. In the public-key setting, $\mathcal{B}$ receives $\mathsf{FE.mpk}$ from its FE challenger, computes $\mathsf{FE.ct}_{\mathsf{id}} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, m_{\mathsf{id}})$ for $\mathsf{id} \in [\mathsf{Bound}_{\mathcal{A}}] \backslash \{w\}$, and sends $\mathsf{MPK} = \mathsf{FE.mpk}$ to $\mathcal{A}$.

- For each function query f_j output by $\mathcal{A}$, $\mathcal{B}$ does the following: $\mathcal{B}$ computes $G_j = G[f_j, s_j, c_{j,w}]$ as in step 5a of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,2}$. $\mathcal{B}$ sends function query G_j to its FE challenger and receives a function key $\mathsf{FE.sk}_{G_j}$. This is a valid function query since for all $j \in [Q]$,

  $$G[f_j, s_j, c_{j,w}](\mathsf{FPFE.msk}_w, \mathsf{PRF}.K_w, 0, 0^{\ell_{\mathsf{SKE.sk}}}) = G[f_j, s_j, c_{j,w}](0^{\ell_{\mathsf{FPFE.msk}}}, 0^{\ell_{\mathsf{PRF}.K}}, 1, \mathsf{SKE.sk})$$

  because $c_{j,w}$ encrypts $(\mathsf{One\text{-}sFE.sk}_{f_j,w}, \mathsf{FPFE.sk}_{H_{j,w}})$ which are generated in the same way as in the $\alpha = 0$ branch of $G[f_j, s_j, c_{j,w}]$. $\mathcal{B}$ then sends $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to $\mathcal{A}$. Note that since $\mathcal{A}$ can only make at most Q function queries, than $\mathcal{B}$ will also make at most Q function queries to its FE challenger.

- For each message query $(x^{(0)}_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i})$ output by $\mathcal{A}$, $\mathcal{B}$ computes $\mathsf{CT}_{\mathsf{id},i}$ as in step 5b of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,2}$ and sends $\mathsf{CT}_{\mathsf{id},i}$ to $\mathcal{A}$.

After $\mathcal{A}$ is done making queries, $\mathcal{A}$ outputs b' which $\mathcal{B}$ also outputs. If the experiment for $\mathcal{A}$ aborts for any reason, $\mathcal{B}$ instead outputs 0. Observe that if $\mathcal{B}$ received FE ciphertexts for the first message in each challenge pair (i.e. either $m_{0,w}$ or m_{id}), then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,1}$, and if $\mathcal{B}$ received FE ciphertexts for the second message in each challenge pair (i.e. either $m_{1,w}$ or m_{id}), then $\mathcal{B}$ emulates $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,2}$. Additionally, $\mathcal{B}$ does not need to know $\mathsf{FE.msk}$ to carry out this experiment and makes at most Q function queries. Thus, by Equation 7, this means that $\mathcal{B}$ breaks the Q-bounded, selective-IND-security of FE as $\mathcal{B}$ can distinguish between the two security games with non-negligible probability. $\square$

$\mathbf{Hybrid}^{\mathcal{A}}_{1,w,3}$: We exchange the randomness generated by $\mathsf{PRF}.K_w$ with true randomness.

This is the same as $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,2}$ except that we change the following steps:

4. **Precompute Values**: For $j \in [Q]$,
   (a) $s_j \leftarrow \{0,1\}^\lambda$.
   (b) $\cancel{(r_{\mathsf{Setup},j,w}, r_{\mathsf{EncSetup},j,w}, r_{\mathsf{KeyGen},j,w}, r_{\mathsf{PRF2},j,w}, r_{H,j,w}) \leftarrow \mathsf{PRF.Eval}(\mathsf{PRF}.K_w, s_j)}$.
   (c) $\mathsf{One\text{-}sFE.msk}_{j,w} \leftarrow \mathsf{One\text{-}sFE.Setup}(1^\lambda; \cancel{r_{\mathsf{Setup},j,w}})$.
   (d) $\mathsf{One\text{-}sFE.Enc.st}_{j,w} \leftarrow \mathsf{One\text{-}sFE.EncSetup}(\mathsf{One\text{-}sFE.msk}_{j,w}; \cancel{r_{\mathsf{EncSetup},j,w}})$.
   (e) $\mathsf{PRF2}.K_{j,w} \leftarrow \mathsf{PRF2.Setup}(1^\lambda; \cancel{r_{\mathsf{PRF2},j,w}})$.
   (f) Let $H_{j,w} = H[\mathsf{One\text{-}sFE.msk}_{j,w}, \mathsf{One\text{-}sFE.Enc.st}_{j,w}, \mathsf{PRF2}.K_{j,w}]$ as defined in Figure 10 (page 50).
   (g) $\mathsf{FPFE.sk}_{H_{j,w}} \leftarrow \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}_w, H_{j,w}; \cancel{r_{H,j,w}})$.

5a. **Function Query:** 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) $\mathsf{One\text{-}sFE.sk}_{f_j,w} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_{j,w}, f_j; \cancel{r_{\mathsf{KeyGen},j,w}})$.
   (b) $c_{j,w} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, (\mathsf{One\text{-}sFE.sk}_{f_j,w}, \mathsf{FPFE.sk}_{H_{j,w}}))$.
   (c) Let $G_j = G[f_j, s_j, c_{j,w}]$ as defined in Figure 9 (page 50).
   (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.

Lemma 5.12. *If* PRF *is a secure PRF, then for all PPT adversaries $\mathcal{A}$ and all $w \in [\mathsf{Bound}_{\mathcal{A}}]$,*

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,2}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,3}(1^\lambda) = 1]\right| \le \mathsf{negl}(\lambda)$$

Proof. Suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$ and a $w \in [\mathsf{Bound}_{\mathcal{A}}]$ such that

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,2}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,3}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{8}$$

We build a PPT adversary $\mathcal{B}$ that breaks the security of PRF. $\mathcal{B}$ first runs steps 1-3 as in $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,3}$ except that $\mathcal{B}$ does not compute $\mathsf{PRF}.K_w$.

For $j \in [Q]$, $\mathcal{B}$ samples $s_j \leftarrow \{0,1\}^\lambda$ and queries its PRF challenger on input s_j to receive values $(r_{j,1}, r_{j,2}, r_{j,3}, r_{j,4}, r_{j,5})$.

$\mathcal{B}$ then runs steps 4-6 as in $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,2}$ except that for each j, instead of computing the values $(r_{\mathsf{Setup},j,w}, r_{\mathsf{EncSetup},j,w}, r_{\mathsf{KeyGen},j,w}, r_{\mathsf{PRF2},j,w}, r_{H,j,w})$ from some PRF key $\mathsf{PRF}.K_w$, $\mathcal{B}$ sets these values equal to $(r_{j,1}, r_{j,2}, r_{j,3}, r_{j,4}, r_{j,5})$.

At the final step, $\mathcal{A}$ outputs b' which $\mathcal{B}$ also outputs. If the experiment for $\mathcal{A}$ aborts for any reason, $\mathcal{B}$ instead outputs 0. Observe that if $\mathcal{B}$'s PRF oracle was a uniform random function R, then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,2}$, and if $\mathcal{B}$'s PRF oracle was $\mathsf{PRF.Eval}(\mathsf{PRF}.K_w, \cdot)$ for some PRF key $\mathsf{PRF}.K_w$, then $\mathcal{B}$ emulates $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,3}$. Additionally, $\mathcal{B}$ does not need to know $\mathsf{PRF}.K_w$ to carry out this experiment. Thus, by Equation 8, this means that $\mathcal{B}$ breaks the security of PRF as $\mathcal{B}$ can distinguish between a random function and PRF evaluations. ◻

Hybrid$^{\mathcal{A}}_{1,w,4,k,0}(1^\lambda)$: Our next goal is to change the ciphertexts for stream identity w from encryptions of $x_w^{(0)}$ to encryptions of $x_w^{(1)}$. We begin this process by changing the behavior of the hybrid one function at a time.

For $j < k$, we set $H_{j,w} = H'[\mathsf{One\text{-}sFE.msk}_k, \mathsf{One\text{-}sFE.Enc.st}_k, \mathsf{PRF2}.K_k]$ which will operate on the second stream input given, namely $x_{w,i}^{(1)}$ which we will additionally encrypt inside $\mathsf{FPFE.ct}_{w,i}$.

For $j = k$, we set $H_{k,w} = H^*$ which simply outputs the last value of its input tuple. To maintain consistency, for each i, we set the last value of the tuple encrypted in $\mathsf{FPFE.ct}_{w,i}$ to $v_{k,w,i}$ which we compute as the output of the original $H_{k,w}$ on $(i, t_{w,i}, x_{w,i}^{(0)}, 0^{\ell_\mathcal{X}}, 0^{\ell_{\mathsf{One\text{-}sFE.ct}}})$. These changes allow us to remove $(\mathsf{One\text{-}sFE.msk}_{k,w}, \mathsf{One\text{-}sFE.Enc.st}_{k,w}, \mathsf{PRF2}.K_{k,w})$ from $\mathsf{FPFE.sk}_{H_{k,w}}$.

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:**
   - *Secret-Key Setting:*
     (a) $\mathsf{SKE.sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
     (b) $\mathsf{FE.msk} \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
     (c) $\mathsf{FE.ek} = \mathsf{FE.msk}$.
   - *Public-Key Setting:*
     (a) $\mathsf{SKE.sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
     (b) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
     (c) $\mathsf{FE.ek} = \mathsf{FE.mpk}$.
     (d) Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.

3. **Encryption Setup:** For $\mathsf{id} \in [\mathsf{Bound}_\mathcal{A}]$ where $\mathsf{Bound}_\mathcal{A}$ is a bound on the runtime of $\mathcal{A}$.
   (a) $\mathsf{PRF}.K_{\mathsf{id}} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
   (b) $\mathsf{FPFE.msk}_{\mathsf{id}} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$.
   (c) If $\mathsf{id} \neq w$, $\mathsf{FE.ct}_{\mathsf{id}} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.ek}, (\mathsf{FPFE.msk}_{\mathsf{id}}, \mathsf{PRF}.K_{\mathsf{id}}, 0, 0^{\ell_{\mathsf{SKE.sk}}}))$.
   (d) If $\mathsf{id} = w$, $\mathsf{FE.ct}_w \leftarrow \mathsf{FE.Enc}(\mathsf{FE.ek}, (0^{\ell_{\mathsf{FPFE.msk}}}, 0^{\ell_{\mathsf{PRF}.K}}, 1, \mathsf{SKE.sk}))$.

4. **Precompute Values**: For $j \in [Q]$,
   (a) $s_j \leftarrow \{0,1\}^\lambda$.
   (b) $\mathsf{One\text{-}sFE.msk}_{j,w} \leftarrow \mathsf{One\text{-}sFE.Setup}(1^\lambda)$.
   (c) $\mathsf{One\text{-}sFE.Enc.st}_{j,w} \leftarrow \mathsf{One\text{-}sFE.EncSetup}(\mathsf{One\text{-}sFE.msk}_{j,w})$.
   (d) $\mathsf{PRF2}.K_{j,w} \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.
   (e) If $j < k$, let $H_{j,w} = H'[\mathsf{One\text{-}sFE.msk}_{j,w}, \mathsf{One\text{-}sFE.Enc.st}_{j,w}, \mathsf{PRF2}.K_{j,w}]$ as defined in Figure 11 (page 51).
   (f) If $j = k$, let $H_{k,w} = H^*$ as defined in Figure 12 (page 51).
   (g) If $j > k$, let $H_{j,w} = H[\mathsf{One\text{-}sFE.msk}_{j,w}, \mathsf{One\text{-}sFE.Enc.st}_{j,w}, \mathsf{PRF2}.K_{j.w}]$ as defined in Figure 10 (page 50).
   (h) $\mathsf{FPFE.sk}_{H_{j,w}} \leftarrow \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}_w, H_{j,w})$.

5. The adversary can make up to Q function queries followed by any polynomial number of message queries.

   (a) **Function Query:** 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:

      i. $\mathsf{One\text{-}sFE.sk}_{f_j,w} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_{j,w}, f_j)$.

      ii. $c_{j,w} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, (\mathsf{One\text{-}sFE.sk}_{f_j,w}, \mathsf{FPFE.sk}_{H_{j,w}}))$.

      iii. Let $G_j = G[f_j, s_j, c_{j,w}]$ as defined in Figure 9 (page 50).

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

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

   (b) **Message Query:** For the i^{th} message query made to stream identity id, $\mathcal{A}$ outputs a message pair $(x^{(0)}_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i})$ where $x^{(0)}_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i} \in \{0,1\}^{\ell_\mathcal{X}}$.

      i. $t_{\mathsf{id},i} \leftarrow \{0,1\}^\lambda$.

      ii. If $\mathsf{id} < w$, $\mathsf{FPFE.ct}_{\mathsf{id},i} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}_\mathsf{id}, (i, t_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i}, 0^{\ell_\mathcal{X}}, 0^{\ell_{\mathsf{One\text{-}sFE.ct}}}))$.

      iii. If $\mathsf{id} = w$,

         A. $r_{k,w,i} \leftarrow \mathsf{PRF2.Eval}(\mathsf{PRF2}.K_{k,w}, t_{w,i})$.

         B. $v_{k,w,i} \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}_{k,w}, \mathsf{One\text{-}sFE.Enc.st}_{k,w}, i, x^{(0)}_{w,i}; r_{k,w,i})$.

         C. $\mathsf{FPFE.ct}_{w,i} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}_w, (i, t_{w,i}, x^{(0)}_{w,i}, x^{(1)}_{w,i}, v_{k,w,i}))$.

      iv. If $\mathsf{id} > w$, $\mathsf{FPFE.ct}_{\mathsf{id},i} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}_\mathsf{id}, (i, t_{\mathsf{id},i}, x^{(0)}_{\mathsf{id},i}, 0^{\ell_\mathcal{X}}, 0^{\ell_{\mathsf{One\text{-}sFE.ct}}}))$.

      v. If $i = 1$, set $\mathsf{CT}_{\mathsf{id},1} = (\mathsf{FE.ct}_\mathsf{id}, \mathsf{FPFE.ct}_{\mathsf{id},1})$. Else, set $\mathsf{CT}_{\mathsf{id},i} = \mathsf{FPFE.ct}_{\mathsf{id},i}$.

      vi. Send $\mathsf{CT}_{\mathsf{id},i}$ to the adversary.

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

Lemma 5.13. *If* FPFE *is Q-bounded, function-private, function-selective-IND-secure, then for all PPT adversaries $\mathcal{A}$ and all $w \in [\mathsf{Bound}_\mathcal{A}]$,*

$$\left|\Pr[\mathbf{Hybrid}^\mathcal{A}_{1,w,3}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^\mathcal{A}_{1,w,4,1,0}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

Proof. Suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$ and a $w \in [\mathsf{Bound}_\mathcal{A}]$ such that

$$\left|\Pr[\mathbf{Hybrid}^\mathcal{A}_{1,w,3}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^\mathcal{A}_{1,w,4,1,0}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{9}$$

We build a PPT adversary $\mathcal{B}$ that breaks the Q-bounded, function-private, function-selective-IND-security of FPFE. $\mathcal{B}$ first runs steps 1-3 as in $\mathbf{Hybrid}^\mathcal{A}_{1,w,3}$ except that $\mathcal{B}$ does not compute $\mathsf{FPFE.msk}_w$.

For $j \in [Q]$, $\mathcal{B}$ does the following: $\mathcal{B}$ computes $(s_j, \mathsf{One\text{-}sFE.msk}_{j,w}, \mathsf{One\text{-}sFE.Enc.st}_{j,w}, \mathsf{PRF2}.K_{j,w})$ as in step 4 of $\mathbf{Hybrid}^\mathcal{A}_{1,w,3}$ and sets $H^{(0)}_{j,w} = H[\mathsf{One\text{-}sFE.msk}_{j,w}, \mathsf{One\text{-}sFE.Enc.st}_{j,w}, \mathsf{PRF2}.K_{j,w}]$.

- If $j = 1$, $\mathcal{B}$ sets its 1^{st} challenge function pair to $(H^{(0)}_{1,w}, H^*)$
- If $j > 1$, $\mathcal{B}$ sets its j^{th} challenge function pair to $(H^{(0)}_{j,w}, H^{(0)}_{j,w})$

$\mathcal{B}$ then sends all Q challenge function pairs to its FPFE challenger and receives $\{\mathsf{FPFE.sk}_{H_{j,w}}\}_{j\in[Q]}$.

- For each function query f_j output by $\mathcal{A}$, $\mathcal{B}$ computes SK_{f_j} as in step 5a of $\mathbf{Hybrid}^\mathcal{A}_{1,w,3}$, and sends SK_{f_j} to $\mathcal{A}$.

- For each message query $(x_{\mathsf{id},i}^{(0)}, x_{\mathsf{id},i}^{(1)})$ output by $\mathcal{A}$:

  If $\mathsf{id} \neq w$, $\mathcal{B}$ computes $\mathsf{CT}_{\mathsf{id},i}$ as in step 5a of $\mathbf{Hybrid}_{1,w,3}^{\mathcal{A}}$, and sends $\mathsf{CT}_{\mathsf{id},i}$ to $\mathcal{A}$.

  If $\mathsf{id} = w$, $\mathcal{B}$ computes $(t_{w,i}, v_{1,w,i})$ as in step 5b of $\mathbf{Hybrid}_{1,w,4,1,0}^{\mathcal{A}}$. $\mathcal{B}$ sends challenge message pair $((i, t_{w,i}, x_{w,i}^{(0)}, 0^{\ell_{\mathcal{X}}}, 0^{\ell_{\mathsf{One\text{-}sFE.ct}}}), (i, t_{w,i}, x_{w,i}^{(0)}, x_{w,i}^{(1)}, v_{1,w,i}))$ to its FPFE challenger and receives $\mathsf{FPFE.ct}_{w,i}$. This is a valid message query since

  - For $j = 1$, $H_{1,w}^{(0)}(i, t_{w,i}, x_{w,i}^{(0)}, 0^{\ell_{\mathcal{X}}}, 0^{\ell_{\mathsf{One\text{-}sFE.ct}}}) = H^*(i, t_{w,i}, x_{w,i}^{(0)}, x_{w,i}^{(1)}, v_{1,w,i})$ since H^* simply outputs $v_{1,w,i}$ which has been programmed to be equal to the lefthand side of the equation
  - For $j \in [Q]\backslash\{1\}$, $H_{j,w}^{(0)}(i, t_{w,i}, x_{w,i}^{(0)}, 0^{\ell_{\mathcal{X}}}, 0^{\ell_{\mathsf{One\text{-}sFE.ct}}}) = H_{j,w}^{(0)}(i, t_{w,i}, x_{w,i}^{(0)}, x_{w,i}^{(1)}, v_{1,w,i})$ since $H_{j,w}^{(0)}$ ignores its last two inputs.

  If $i = 1$, $\mathcal{B}$ sets $\mathsf{CT}_{w,1} = (\mathsf{FE.ct}_w, \mathsf{FPFE.ct}_{w,1})$. Else, $\mathcal{B}$ sets $\mathsf{CT}_{w,i} = \mathsf{FPFE.ct}_{w,i}$. $\mathcal{B}$ sends $\mathsf{CT}_{w,i}$ to $\mathcal{A}$.

After $\mathcal{A}$ is done making queries, $\mathcal{A}$ outputs b' which $\mathcal{B}$ also outputs. If the experiment for $\mathcal{A}$ aborts for any reason, $\mathcal{B}$ instead outputs 0. 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 $\mathbf{Hybrid}_{1,w,3}^{\mathcal{A}}$, 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 $\mathbf{Hybrid}_{1,w,4,1,0}^{\mathcal{A}}$. Additionally, $\mathcal{B}$ does not need to know $\mathsf{FPFE.msk}_w$ to carry out this experiment and makes only Q function queries. Thus, by Equation 9, this means that $\mathcal{B}$ breaks the Q-bounded, function-private, function-selective-IND-security of FPFE as $\mathcal{B}$ can distinguish between the two security games with non-negligible probability. $\square$

$\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,1}(1^\lambda)$: We exchange the randomness generated by $\mathsf{PRF2}.K_{k,w}$ with true randomness. This is the same as $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,0}$ except that we change the following steps:

5b. **Message Query:** For the i^{th} message query made to stream identity id, $\mathcal{A}$ outputs a message pair $(x^{(0)}_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i})$ where $x^{(0)}_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i} \in \{0,1\}^{\ell_\mathcal{X}}$.

(a) $t_{\mathsf{id},i} \leftarrow \{0,1\}^\lambda$.

(b) If $\mathsf{id} < w$, $\mathsf{FPFE.ct}_{\mathsf{id},i} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}_{\mathsf{id}}, (i, t_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i}, 0^{\ell_\mathcal{X}}, 0^{\ell_{\mathsf{One\text{-}sFE.ct}}}))$.

(c) If $\mathsf{id} = w$,

i. $\cancel{r_{k,w,i} \leftarrow \mathsf{PRF2.Eval}(\mathsf{PRF2}.K_{k,w}, t_{w,i})}$.

ii. $v_{k,w,i} \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}_{k,w}, \mathsf{One\text{-}sFE.Enc.st}_{k,w}, i, x^{(0)}_{w,i}; \cancel{r_{k,w,i}})$.

iii. $\mathsf{FPFE.ct}_{w,i} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}_w, (i, t_{w,i}, x^{(0)}_{w,i}, x^{(1)}_{w,i}, v_{k,w,i}))$.

(d) If $\mathsf{id} > w$, $\mathsf{FPFE.ct}_{\mathsf{id},i} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}_{\mathsf{id}}, (i, t_{\mathsf{id},i}, x^{(0)}_{\mathsf{id},i}, 0^{\ell_\mathcal{X}}, 0^{\ell_{\mathsf{One\text{-}sFE.ct}}}))$.

(e) If $i = 1$, set $\mathsf{CT}_{\mathsf{id},1} = (\mathsf{FE.ct}_{\mathsf{id}}, \mathsf{FPFE.ct}_{\mathsf{id},1})$. Else, set $\mathsf{CT}_{\mathsf{id},i} = \mathsf{FPFE.ct}_{\mathsf{id},i}$.

(f) Send $\mathsf{CT}_{\mathsf{id},i}$ to the adversary.

Lemma 5.14. *If* $\mathsf{PRF2}$ *is a secure PRF, then for all PPT adversaries* $\mathcal{A}$*, all* $w \in [\mathsf{Bound}_\mathcal{A}]$*, and all* $k \in [Q]$*,*

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,0}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,1}(1^\lambda) = 1]\right| \le \mathsf{negl}(\lambda)$$

Proof. Suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$, $w \in [\mathsf{Bound}_\mathcal{A}]$, and $k \in [Q]$ such that

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,0}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,1}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{10}$$

We build a PPT adversary $\mathcal{B}$ that breaks the security of $\mathsf{PRF2}$. $\mathcal{B}$ first runs steps 1-4 as in $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,1}$ except that $\mathcal{B}$ does not compute $\mathsf{PRF2}.K_{k,w}$.

- For each function query f_j output by $\mathcal{A}$, $\mathcal{B}$ computes SK_{f_j} as in step 5a of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,1}$, and sends SK_{f_j} to $\mathcal{A}$.

- For each message query $(x^{(0)}_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i})$ output by $\mathcal{A}$, $\mathcal{B}$ computes $\mathsf{CT}_{\mathsf{id},i}$ as in step 5b of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,1}$ except that if $\mathsf{id} = w$, instead of computing $r_{k,w,i}$ using $\mathsf{PRF2}$, $\mathcal{B}$ samples $t_{w,i} \leftarrow \{0,1\}^\lambda$, sends $t_{w,i}$ to its $\mathsf{PRF2}$ challenger to receiver $r^*_{w,i}$ and sets $r_{k,w,i} = r^*_{w,i}$. $\mathcal{B}$ then sends $\mathsf{CT}_{\mathsf{id},i}$ to $\mathcal{A}$.

After $\mathcal{A}$ is done making queries, $\mathcal{A}$ outputs b' which $\mathcal{B}$ also outputs. If the experiment for $\mathcal{A}$ aborts for any reason, $\mathcal{B}$ instead outputs 0. Observe that if $\mathcal{B}$'s $\mathsf{PRF2}$ oracle was a uniform random function R, then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,1}$, and if $\mathcal{B}$'s $\mathsf{PRF2}$ oracle was $\mathsf{PRF2.Eval}(\mathsf{PRF2}.K_{k,w}, \cdot)$ for some $\mathsf{PRF2}$ key $\mathsf{PRF2}.K_{k,w}$, then $\mathcal{B}$ emulates $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,0}$. Additionally, $\mathcal{B}$ does not need to know $\mathsf{PRF2}.K_{k,w}$ to carry out this experiment. Thus, by Equation 10, this means that $\mathcal{B}$ breaks the security of $\mathsf{PRF2}$ as $\mathcal{B}$ can distinguish between a random function and $\mathsf{PRF2}$ evaluations. $\square$

$\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,2}(1^\lambda)$: We now invoke the security of $\mathsf{One\text{-}sFE}$ to change each $v_{k,w,i}$ from an encryption of $x^{(0)}_{w,i}$ to an encryption of $x^{(1)}_{w,i}$.

This is the same as $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,1}$ except that we change the following steps:

5b **Message Query:** For the i^{th} message query made to stream identity id, $\mathcal{A}$ outputs a message pair $(x^{(0)}_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i})$ where $x^{(0)}_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i} \in \{0,1\}^{\ell_{\mathcal{X}}}$.

(a) $t_{\mathsf{id},i} \leftarrow \{0,1\}^\lambda$.

(b) If $\mathsf{id} < w$, $\mathsf{FPFE.ct}_{\mathsf{id},i} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}_{\mathsf{id}}, (i, t_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i}, 0^{\ell_{\mathcal{X}}}, 0^{\ell_{\mathsf{One\text{-}sFE.ct}}}))$.

(c) If $\mathsf{id} = w$,

i. $v_{k,w,i} \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}_{k,w}, \mathsf{One\text{-}sFE.Enc.st}_{k,w}, i, x^{(1)}_{w,i})$.

ii. $\mathsf{FPFE.ct}_{w,i} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}_w, (i, t_{w,i}, x^{(0)}_{w,i}, x^{(1)}_{w,i}, v_{k,w,i}))$.

(d) If $\mathsf{id} > w$, $\mathsf{FPFE.ct}_{\mathsf{id},i} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}_{\mathsf{id}}, (i, t_{\mathsf{id},i}, x^{(0)}_{\mathsf{id},i}, 0^{\ell_{\mathcal{X}}}, 0^{\ell_{\mathsf{One\text{-}sFE.ct}}}))$.

(e) If $i = 1$, set $\mathsf{CT}_{\mathsf{id},1} = (\mathsf{FE.ct}_{\mathsf{id}}, \mathsf{FPFE.ct}_{\mathsf{id},1})$. Else, set $\mathsf{CT}_{\mathsf{id},i} = \mathsf{FPFE.ct}_{\mathsf{id},i}$.

(f) Send $\mathsf{CT}_{\mathsf{id},i}$ to the adversary.

Lemma 5.15. *If* $\mathsf{One\text{-}sFE}$ *is single-key, single-ciphertext, function-selective-IND-secure, then for all PPT adversaries* $\mathcal{A}$*, all* $w \in [\mathsf{Bound}_{\mathcal{A}}]$*, and all* $k \in [Q]$*,*

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,1}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,2}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

Proof. Suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$, $w \in [\mathsf{Bound}_{\mathcal{A}}]$, and $k \in [Q]$ such that

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,1}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,2}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{11}$$

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 steps as in 1-4 of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,2}$ except that $\mathcal{B}$ does not compute $(\mathsf{One\text{-}sFE.msk}_{k,w}, \mathsf{One\text{-}sFE.Enc.st}_{k,w})$ in step 4. Note that these values are not needed to compute these steps since $H_{k,w} = H^*$.

- For each function query f_j output by $\mathcal{A}$:

  If $j \neq k$, $\mathcal{B}$ computes SK_{f_j} as in step 5a of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,2}$, and sends SK_{f_j} to $\mathcal{A}$.

  If $j = k$, $\mathcal{B}$ sends f_k to its $\mathsf{One\text{-}sFE}$ challenger and receives a function key $\mathsf{One\text{-}sFE.sk}_{f_k}$. $\mathcal{B}$ computes $c_{k,w} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, (\mathsf{One\text{-}sFE.sk}_{f_k}, \mathsf{FPFE.sk}_{H_{k,w}}))$ and $\mathsf{FE.sk}_{G_k} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_k)$ for $G_k = G[f_k, s_k, c_{k,w}]$. $\mathcal{B}$ sends $\mathsf{SK}_{f_k} = \mathsf{FE.sk}_{G_k}$ to $\mathcal{A}$.

- For each message query $(x^{(0)}_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i})$ output by $\mathcal{A}$:

  If $\mathsf{id} \neq w$, $\mathcal{B}$ computes $\mathsf{CT}_{\mathsf{id},i}$ as in step 5b of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,2}$, and sends $\mathsf{CT}_{\mathsf{id},i}$ to $\mathcal{A}$.

  If $\mathsf{id} = w$, $\mathcal{B}$ sends challenge message pair $(x^{(0)}_{w,i}, x^{(1)}_{w,i})$ to its $\mathsf{One\text{-}sFE}$ challenger and receives $\mathsf{One\text{-}sFE.ct}_{w,i}$. This is a valid message query since $\mathcal{A}$ is restricted to function and message queries that satisfy

  $$f_k(x^{(0)}_w) = f_k(x^{(1)}_w)$$

  $\mathcal{B}$ then computes $t_{w,i} \leftarrow \{0,1\}^\lambda$ and $\mathsf{FPFE.ct}_{w,i} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}_w, (i, t_{w,i}, x^{(0)}_{w,i}, x^{(1)}_{w,i}, \mathsf{One\text{-}sFE.ct}_{w,i})$. If $i = 1$, $\mathcal{B}$ sets $\mathsf{CT}_{w,1} = (\mathsf{FE.ct}_w, \mathsf{FPFE.ct}_{w,1})$. Else, $\mathcal{B}$ sets $\mathsf{CT}_{w,i} = \mathsf{FPFE.ct}_{w,i}$. $\mathcal{B}$ sends $\mathsf{CT}_{w,i}$ to $\mathcal{A}$.

After $\mathcal{A}$ is done making queries, $\mathcal{A}$ outputs b' which $\mathcal{B}$ also outputs. If the experiment for $\mathcal{A}$ aborts for any reason, $\mathcal{B}$ instead outputs 0. Observe that if $\mathcal{B}$ received ciphertexts for $x_w^{(0)}$ from its $\mathsf{One\text{-}sFE}$ challenger, then $\mathcal{B}$ exactly emulates $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,1}$, and if $\mathcal{B}$ received ciphertexts for $x_w^{(1)}$, then $\mathcal{B}$ emulates $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,2}$. Additionally, $\mathcal{B}$ does not need to know $(\mathsf{One\text{-}sFE.msk}_{k,w}, \mathsf{One\text{-}sFE.Enc.st}_{k,w})$ to carry out this experiment and makes only one function query followed by one message query to its $\mathsf{One\text{-}sFE}$ challenger. Thus, by Equation 11, 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 security games with non-negligible probability. $\square$

$\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,3}(1^\lambda)$: We revert back to using $\mathsf{PRF2}.K_{k,w}$ to compute the randomness needed for determining each $v_{k,w,i}$.

This is the same as $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,2}$ except that we change the following steps:

5b. **Message Query:** For the i^{th} message query made to stream identity id, $\mathcal{A}$ outputs a message pair $(x^{(0)}_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i})$ where $x^{(0)}_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i} \in \{0,1\}^{\ell_{\mathcal{X}}}$.

(a) $t_{\mathsf{id},i} \leftarrow \{0,1\}^\lambda$.

(b) If $\mathsf{id} < w$, $\mathsf{FPFE.ct}_{\mathsf{id},i} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}_{\mathsf{id}}, (i, t_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i}, 0^{\ell_{\mathcal{X}}}, 0^{\ell_{\mathsf{One\text{-}sFE.ct}}}))$.

(c) If $\mathsf{id} = w$,

i. $r_{k,w,i} \leftarrow \mathsf{PRF2.Eval}(\mathsf{PRF2}.K_{k,w}, t_{w,i})$.

ii. $v_{k,w,i} \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}_{k,w}, \mathsf{One\text{-}sFE.Enc.st}_{k,w}, i, x^{(1)}_{w,i}; r_{k,w,i})$.

iii. $\mathsf{FPFE.ct}_{w,i} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}_w, (i, t_{w,i}, x^{(0)}_{w,i}, x^{(1)}_{w,i}, v_{k,w,i}))$.

(d) If $\mathsf{id} > w$, $\mathsf{FPFE.ct}_{\mathsf{id},i} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}_{\mathsf{id}}, (i, t_{\mathsf{id},i}, x^{(0)}_{\mathsf{id},i}, 0^{\ell_{\mathcal{X}}}, 0^{\ell_{\mathsf{One\text{-}sFE.ct}}}))$.

(e) If $i = 1$, set $\mathsf{CT}_{\mathsf{id},1} = (\mathsf{FE.ct}_{\mathsf{id}}, \mathsf{FPFE.ct}_{\mathsf{id},1})$. Else, set $\mathsf{CT}_{\mathsf{id},i} = \mathsf{FPFE.ct}_{\mathsf{id},i}$.

(f) Send $\mathsf{CT}_{\mathsf{id},i}$ to the adversary.

Lemma 5.16. *If* $\mathsf{PRF2}$ *is a secure PRF, then for all PPT adversaries* $\mathcal{A}$*, all* $w \in [\mathsf{Bound}_{\mathcal{A}}]$*, and all* $k \in [Q]$*,*

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,2}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,3}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

Proof. This proof is essentially the same as the proof of Lemma 5.14. □

$\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,4}(1^\lambda)$: We change function $H_{k,w}$ from H^* to $H'[\mathsf{One\text{-}sFE.msk}_{k,w}, \mathsf{One\text{-}sFE.Enc.st}_{k,w}, \mathsf{PRF2}.K_{k,w}]$, which operates on the second stream input given, namely $x^{(1)}_{w,i}$.

This is the same as $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,3}$ except that we change the following steps:

4. **Precompute Values**: For $j \in [Q]$,
   (a) $s_j \leftarrow \{0,1\}^\lambda$.
   (b) $\mathsf{One\text{-}sFE.msk}_{j,w} \leftarrow \mathsf{One\text{-}sFE.Setup}(1^\lambda)$.
   (c) $\mathsf{One\text{-}sFE.Enc.st}_{j,w} \leftarrow \mathsf{One\text{-}sFE.EncSetup}(\mathsf{One\text{-}sFE.msk}_{j,w})$.
   (d) $\mathsf{PRF2}.K_{j,w} \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.
   (e) If $j \leq k$, let $H_{j,w} = H'[\mathsf{One\text{-}sFE.msk}_{j,w}, \mathsf{One\text{-}sFE.Enc.st}_{j,w}, \mathsf{PRF2}.K_{j,w}]$ as defined in Figure 11 (page 51).
   (f) ~~If $j = k$, let $H_{k,w} = H^*$ as defined in Figure 12 (page 51).~~
   (g) If $j > k$, let $H_{j,w} = H[\mathsf{One\text{-}sFE.msk}_{j,w}, \mathsf{One\text{-}sFE.Enc.st}_{j,w}, \mathsf{PRF2}.K_{j.w}]$ as defined in Figure 10 (page 50).
   (h) $\mathsf{FPFE.sk}_{H_{j,w}} \leftarrow \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}_w, H_{j,w})$.

Lemma 5.17. *If* FPFE *is Q-bounded, function-private, function-selective-IND-secure, then for all PPT adversaries $\mathcal{A}$, all $w \in [\mathsf{Bound}_{\mathcal{A}}]$, and all $k \in [Q]$,*

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,3}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,4}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

Proof. Suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$, $w \in [\mathsf{Bound}_{\mathcal{A}}]$, and $k \in [Q]$ such that

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,3}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,4}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{12}$$

We build a PPT adversary $\mathcal{B}$ that breaks the Q-bounded, function-private, function-selective-IND-security of FPFE. $\mathcal{B}$ first runs steps 1-3 of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,4}$ except that $\mathcal{B}$ does not compute $\mathsf{FPFE.msk}_w$.

For $j \in [Q]$, $\mathcal{B}$ does the following: $\mathcal{B}$ computes $(s_j, \mathsf{One\text{-}sFE.msk}_{j,w}, \mathsf{One\text{-}sFE.Enc.st}_{j,w}, \mathsf{PRF2}.K_{j,w})$ as in step 4 of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,4}$. $\mathcal{B}$ sets $H^{(0)}_{j,w} = H[\mathsf{One\text{-}sFE.msk}_{j,w}, \mathsf{One\text{-}sFE.Enc.st}_{j,w}, \mathsf{PRF2}.K_{j,w}]$ and $H^{(1)}_{j,w} = H'[\mathsf{One\text{-}sFE.msk}_{j,w}, \mathsf{One\text{-}sFE.Enc.st}_{j,w}, \mathsf{PRF2}.K_{j,w}]$.

- If $j < k$, $\mathcal{B}$ sets its j^{th} challenge function pair to $(H^{(1)}_{j,w}, H^{(1)}_{j,w})$.
- If $j = k$, $\mathcal{B}$ sets its j^{th} challenge function pair to $(H^*, H^{(1)}_{k,w})$.
- If $j > k$, $\mathcal{B}$ sets its j^{th} challenge function pair to $(H^{(0)}_{j,w}, H^{(0)}_{j,w})$.

$\mathcal{B}$ then sends all Q challenge function pairs to its FPFE challenger and receives $\{\mathsf{FPFE.sk}_{H_{j,w}}\}_{j\in[Q]}$.

- For each function query f_j output by $\mathcal{A}$, $\mathcal{B}$ computes SK_{f_j} as in step 5a of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,4}$, and sends SK_{f_j} to $\mathcal{A}$.

- For each message query $(x^{(0)}_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i})$ output by $\mathcal{A}$:

  If $\mathsf{id} \neq w$, $\mathcal{B}$ computes $\mathsf{CT}_{\mathsf{id},i}$ as in step 5b of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,4}$, and sends $\mathsf{CT}_{\mathsf{id},i}$ to $\mathcal{A}$.

  If $\mathsf{id} = w$, $\mathcal{B}$ computes $(t_{w,i}, v_{k,w,i})$ as in step 5b of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,4}$. $\mathcal{B}$ sends challenge message pair $((i, t_{w,i}, x^{(0)}_{w,i}, x^{(1)}_{w,i}, v_{k,w,i}), (i, t_{w,i}, x^{(0)}_{w,i}, x^{(1)}_{w,i}, v_{k,w,i}))$ to its FPFE challenger and receives $\mathsf{FPFE.ct}_{w,i}$. This is a valid message query since

  - For $j = k$, $H^*(i, t_{w,i}, x^{(0)}_{w,i}, x^{(1)}_{w,i}, v_{k,w,i}) = H^{(1)}_{k,w}(i, t_{w,i}, x^{(0)}_{w,i}, x^{(1)}_{w,i}, v_{k,w,i}))$ since H^* simply outputs $v_{k,w,i}$ which has been programmed to be equal to the righthand side of the equation
  - For $j \in [Q]\backslash\{k\}$, the functions and messages in each function or message pair are the same.

  If $i = 1$, $\mathcal{B}$ sets $\mathsf{CT}_{w,1} = (\mathsf{FE.ct}_w, \mathsf{FPFE.ct}_{w,1})$. Else, $\mathcal{B}$ sets $\mathsf{CT}_{w,i} = \mathsf{FPFE.ct}_{w,i}$. $\mathcal{B}$ sends $\mathsf{CT}_{w,i}$ to $\mathcal{A}$.

After $\mathcal{A}$ is done making queries, $\mathcal{A}$ outputs b' which $\mathcal{B}$ also outputs. If the experiment for $\mathcal{A}$ aborts for any reason, $\mathcal{B}$ instead outputs 0. 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 $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,3}$, 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 $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,4}$. Additionally, $\mathcal{B}$ does not need to know $\mathsf{FPFE.msk}_w$ to carry out this experiment and makes only Q function queries. Thus, by Equation 12, this means that $\mathcal{B}$ breaks the Q-bounded, function-private, function-selective-IND-security of FPFE as $\mathcal{B}$ can distinguish between the two security games with non-negligible probability. $\square$

Lemma 5.18. *If* FPFE *is Q-bounded, function-private, function-selective-IND-secure, then for all PPT adversaries $\mathcal{A}$, all $w \in [\mathsf{Bound}_{\mathcal{A}}]$, and all $k \in [Q-1]$,*

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,4}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k+1,0}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

Proof. Suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$, $w \in [\mathsf{Bound}_{\mathcal{A}}]$, and $k \in [Q-1]$ such that

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,4}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k+1,0}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{13}$$

We build a PPT adversary $\mathcal{B}$ that breaks the Q-bounded, function-private, function-selective-IND-security of FPFE. $\mathcal{B}$ first runs steps 1-3 of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,4}$ except that $\mathcal{B}$ does not compute $\mathsf{FPFE.msk}_w$.

For $j \in [Q]$, $\mathcal{B}$ does the following: $\mathcal{B}$ computes $(s_j, \mathsf{One\text{-}sFE.msk}_{j,w}, \mathsf{One\text{-}sFE.Enc.st}_{j,w}, \mathsf{PRF2}.K_{j,w})$ as in step 4 of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,4}$. $\mathcal{B}$ sets $H^{(0)}_{j,w} = H[\mathsf{One\text{-}sFE.msk}_{j,w}, \mathsf{One\text{-}sFE.Enc.st}_{j,w}, \mathsf{PRF2}.K_{j,w}]$ and $H^{(1)}_{j,w} = H'[\mathsf{One\text{-}sFE.msk}_{j,w}, \mathsf{One\text{-}sFE.Enc.st}_{j,w}, \mathsf{PRF2}.K_{j,w}]$.

- If $j < k+1$, $\mathcal{B}$ sets its j^{th} challenge function pair to $(H^{(1)}_{j,w}, H^{(1)}_{j,w})$.
- If $j = k+1$, $\mathcal{B}$ sets its j^{th} challenge function pair to $(H^{(0)}_{k+1,w}, H^*)$.
- If $j > k+1$, $\mathcal{B}$ sets its j^{th} challenge function pair to $(H^{(0)}_{j,w}, H^{(0)}_{j,w})$.

$\mathcal{B}$ then sends all Q challenge function pairs to its FPFE challenger and receives $\{\mathsf{FPFE.sk}_{H_{j,w}}\}_{j\in[Q]}$.

- For each function query f_j output by $\mathcal{A}$, $\mathcal{B}$ computes SK_{f_j} as in step 5a of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,4}$, and sends SK_{f_j} to $\mathcal{A}$.

- For each message query $(x^{(0)}_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i})$ output by $\mathcal{A}$:

  If $\mathsf{id} \neq w$, $\mathcal{B}$ computes $\mathsf{CT}_{\mathsf{id},i}$ as in step 5b of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,1,4}$, and sends $\mathsf{CT}_{\mathsf{id},i}$ to $\mathcal{A}$.

  If $\mathsf{id} = w$, $\mathcal{B}$ samples $t_{w,i} \leftarrow \{0,1\}^\lambda$ and uses this to compute $v_{k,w,i}$ as in step 5b of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,4}$ and $v_{k+1,w,i}$ as in step 5b of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k+1,0}$. $\mathcal{B}$ sends challenge message pair $((i, t_{w,i}, x^{(0)}_{w,i}, x^{(1)}_{w,i}, v_{k,w,i}), (i, t_{w,i}, x^{(0)}_{w,i}, x^{(1)}_{w,i}, v_{k+1,w,i}))$ to its FPFE challenger and receives $\mathsf{FPFE.ct}_{w,i}$. This is a valid message query since

  – For $j = k+1$, $H^{(0)}_{k+1,w}(i, t_{w,i}, x^{(0)}_{w,i}, x^{(1)}_{w,i}, v_{k,w,i}) = H^*(i, t_{w,i}, x^{(0)}_{w,i}, x^{(1)}_{w,i}, v_{k+1,w,i}))$ since H^* simply outputs $v_{k+1,w,i}$ which has been programmed to be equal to the lefthand side of the equation

  – For $j \in [Q] \backslash \{k+1\}$,

$$H^{(0)}_{j,w}(i, t_{w,i}, x^{(0)}_{w,i}, x^{(1)}_{w,i}, v_{k,w,i}) = H^{(0)}_{j,w}(i, t_{w,i}, x^{(0)}_{w,i}, x^{(1)}_{w,i}, v_{k+1,w,i}))$$

$$H^{(1)}_{j,w}(i, t_{w,i}, x^{(0)}_{w,i}, x^{(1)}_{w,i}, v_{k,w,i}) = H^{(1)}_{j,w}(i, t_{w,i}, x^{(0)}_{w,i}, x^{(1)}_{w,i}, v_{k+1,w,i}))$$

  since $H^{(0)}_{j,w}$ and $H^{(1)}_{j,w}$ ignore the last input.

  If $i = 1$, $\mathcal{B}$ sets $\mathsf{CT}_{w,1} = (\mathsf{FE.ct}_w, \mathsf{FPFE.ct}_{w,1})$. Else, $\mathcal{B}$ sets $\mathsf{CT}_{w,i} = \mathsf{FPFE.ct}_{w,i}$. $\mathcal{B}$ sends $\mathsf{CT}_{w,i}$ to $\mathcal{A}$.

After $\mathcal{A}$ is done making queries, $\mathcal{A}$ outputs b' which $\mathcal{B}$ also outputs. If the experiment for $\mathcal{A}$ aborts for any reason, $\mathcal{B}$ instead outputs 0. 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 $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k,4}$, 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 $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,k+1,0}$. Additionally, $\mathcal{B}$ does not need to know $\mathsf{FPFE.msk}_w$ to carry out this experiment and makes only Q function queries. Thus, by Equation 13, this means that $\mathcal{B}$ breaks the Q-bounded, function-private, function-selective-IND-security of FPFE as $\mathcal{B}$ can distinguish between the two security games with non-negligible probability. □

$\mathbf{Hybrid}^{\mathcal{A}}_{1,w,5}(1^\lambda)$: This is identical to $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,Q,4}$. Observe that we have now set each $H_{j,w}$ to $H'[\mathsf{One\text{-}sFE.msk}_{j,w}, \mathsf{One\text{-}sFE.Enc.st}_{j,w}, \mathsf{PRF2}.K_{j,w}]$ which means that every function key will compute its output values for stream identity w using $x_w^{(1)}$ instead of $x_w^{(0)}$.

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:**

   - *Secret-Key Setting:*
     (a) $\mathsf{SKE.sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
     (b) $\mathsf{FE.msk} \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
     (c) $\mathsf{FE.ek} = \mathsf{FE.msk}$.
   - *Public-Key Setting:*
     (a) $\mathsf{SKE.sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
     (b) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
     (c) $\mathsf{FE.ek} = \mathsf{FE.mpk}$.
     (d) Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.

3. **Encryption Setup:** For $\mathsf{id} \in [\mathsf{Bound}_{\mathcal{A}}]$ where $\mathsf{Bound}_{\mathcal{A}}$ is a bound on the runtime of $\mathcal{A}$.

   (a) $\mathsf{PRF}.K_{\mathsf{id}} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
   (b) $\mathsf{FPFE.msk}_{\mathsf{id}} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$.
   (c) If $\mathsf{id} \neq w$, $\mathsf{FE.ct}_{\mathsf{id}} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.ek}, (\mathsf{FPFE.msk}_{\mathsf{id}}, \mathsf{PRF}.K_{\mathsf{id}}, 0, 0^{\ell_{\mathsf{SKE.sk}}}))$.
   (d) If $\mathsf{id} = w$, $\mathsf{FE.ct}_w \leftarrow \mathsf{FE.Enc}(\mathsf{FE.ek}, (0^{\ell_{\mathsf{FPFE.msk}}}, 0^{\ell_{\mathsf{PRF}.K}}, 1, \mathsf{SKE.sk}))$.

4. **Precompute Values**: For $j \in [Q]$,

   (a) $s_j \leftarrow \{0,1\}^\lambda$.
   (b) $\mathsf{One\text{-}sFE.msk}_{j,w} \leftarrow \mathsf{One\text{-}sFE.Setup}(1^\lambda)$.
   (c) $\mathsf{One\text{-}sFE.Enc.st}_{j,w} \leftarrow \mathsf{One\text{-}sFE.EncSetup}(\mathsf{One\text{-}sFE.msk}_{j,w})$.
   (d) $\mathsf{PRF2}.K_{j,w} \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.
   (e) Let $H_{j,w} = H'[\mathsf{One\text{-}sFE.msk}_{j,w}, \mathsf{One\text{-}sFE.Enc.st}_{j,w}, \mathsf{PRF2}.K_{j,w}]$ as defined in Figure 11 (page 51).
   (f) $\mathsf{FPFE.sk}_{H_{j,w}} \leftarrow \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}_w, H_{j,w})$.

5. The adversary can make up to Q function queries followed by any polynomial number of message queries.

   (a) **Function Query:** 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:
      i. $\mathsf{One\text{-}sFE.sk}_{f_j,w} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_{j,w}, f_j)$.
      ii. $c_{j,w} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, (\mathsf{One\text{-}sFE.sk}_{f_j,w}, \mathsf{FPFE.sk}_{H_{j,w}}))$.
      iii. Let $G_j = G[f_j, s_j, c_{j,w}]$ as defined in Figure 9 (page 50).
      iv. $\mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_j)$.
      v. Send $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to the adversary.

(b) **Message Query:** For the i^{th} message query made to stream identity id, $\mathcal{A}$ outputs a message pair $(x^{(0)}_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i})$ where $x^{(0)}_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i} \in \{0,1\}^{\ell_{\mathcal{X}}}$.

i. $t_{\mathsf{id},i} \leftarrow \{0,1\}^{\lambda}$.

ii. If $\mathsf{id} < w$, $\mathsf{FPFE.ct}_{\mathsf{id},i} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}_{\mathsf{id}}, (i, t_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i}, 0^{\ell_{\mathcal{X}}}, 0^{\ell_{\mathsf{One\text{-}sFE.ct}}}))$.

iii. If $\mathsf{id} = w$,

A. $r_{Q,w,i} \leftarrow \mathsf{PRF2.Eval}(\mathsf{PRF2}.K_{Q,w}, t_{w,i})$.

B. $v_{Q,w,i} \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}_{Q,w}, \mathsf{One\text{-}sFE.Enc.st}_{Q,w}, i, x^{(1)}_{w,i}; r_{Q,w,i})$.

C. $\mathsf{FPFE.ct}_{w,i} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}_{w}, (i, t_{w,i}, x^{(0)}_{w,i}, x^{(1)}_{w,i}, v_{Q,w,i}))$.

iv. If $\mathsf{id} > w$, $\mathsf{FPFE.ct}_{\mathsf{id},i} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}_{\mathsf{id}}, (i, t_{\mathsf{id},i}, x^{(0)}_{\mathsf{id},i}, 0^{\ell_{\mathcal{X}}}, 0^{\ell_{\mathsf{One\text{-}sFE.ct}}}))$.

v. If $i = 1$, set $\mathsf{CT}_{\mathsf{id},1} = (\mathsf{FE.ct}_{\mathsf{id}}, \mathsf{FPFE.ct}_{\mathsf{id},1})$. Else, set $\mathsf{CT}_{\mathsf{id},i} = \mathsf{FPFE.ct}_{\mathsf{id},i}$.

vi. Send $\mathsf{CT}_{\mathsf{id},i}$ to the adversary.

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

Lemma 5.19. *For all adversaries $\mathcal{A}$ and all $w \in [\mathsf{Bound}_{\mathcal{A}}]$,*

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,4,Q,4}(1^{\lambda}) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,5}(1^{\lambda}) = 1]\right| = 0$$

Proof. The hybrids are identical. □

$\mathbf{Hybrid}^{\mathcal{A}}_{1,w,6}(1^\lambda)$: Since every function key now computes its output values for stream identity w using $x_w^{(1)}$ instead of $x_w^{(0)}$, we can fully switch to using $x_w^{(1)}$. For every i, we change $\mathsf{FPFE.ct}_{w,i}$ to an encryption of $(i, t_{w,i}, x_{w,i}^{(1)}, 0^{\ell_{\mathcal{X}}}, 0^{\ell_{\mathsf{One\text{-}sFE.ct}}})$, and for every j, we restore $H_{j,w}$ to its original value.

This is the same as $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,5}$ except that we change the following steps:

4. **Precompute Values**: For $j \in [Q]$,
   (a) $s_j \leftarrow \{0,1\}^\lambda$.
   (b) $\mathsf{One\text{-}sFE.msk}_{j,w} \leftarrow \mathsf{One\text{-}sFE.Setup}(1^\lambda)$.
   (c) $\mathsf{One\text{-}sFE.Enc.st}_{j,w} \leftarrow \mathsf{One\text{-}sFE.EncSetup}(\mathsf{One\text{-}sFE.msk}_{j,w})$.
   (d) $\mathsf{PRF2}.K_{j,w} \leftarrow \mathsf{PRF2.Setup}(1^\lambda)$.
   (e) Let $H_{j,w} = H[\mathsf{One\text{-}sFE.msk}_{j,w}, \mathsf{One\text{-}sFE.Enc.st}_{j,w}, \mathsf{PRF2}.K_{j,w}]$ as defined in Figure 10 (page 50).
   (f) $\mathsf{FPFE.sk}_{H_{j,w}} = \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}_w, H_{j,w})$.

5b. **Message Query:** For the i^{th} message query made to stream identity id, $\mathcal{A}$ outputs a message pair $(x_{\mathsf{id},i}^{(0)}, x_{\mathsf{id},i}^{(1)})$ where $x_{\mathsf{id},i}^{(0)}, x_{\mathsf{id},i}^{(1)} \in \{0,1\}^{\ell_{\mathcal{X}}}$.
   (a) $t_{\mathsf{id},i} \leftarrow \{0,1\}^\lambda$.
   (b) If $\mathsf{id} \leq w$, $\mathsf{FPFE.ct}_{\mathsf{id},i} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}_{\mathsf{id}}, (i, t_{\mathsf{id},i}, x_{\mathsf{id},i}^{(1)}, 0^{\ell_{\mathcal{X}}}, 0^{\ell_{\mathsf{One\text{-}sFE.ct}}}))$.
   (c) ~~If $\mathsf{id} = w$,~~
      i. ~~$r_{Q,w,i} \leftarrow \mathsf{PRF2.Eval}(\mathsf{PRF2}.K_{Q,w}, t_{w,i})$.~~
      ii. ~~$v_{Q,w,i} \leftarrow \mathsf{One\text{-}sFE.Enc}(\mathsf{One\text{-}sFE.msk}_{Q,w}, \mathsf{One\text{-}sFE.Enc.st}_{Q,w}, i, x_{w,i}^{(1)}; r_{Q,w,i})$.~~
      iii. ~~$\mathsf{FPFE.ct}_{w,i} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}_w, (i, t_{w,i}, x_{w,i}^{(0)}, x_{w,i}^{(1)}, v_{Q,w,i}))$.~~
   (d) If $\mathsf{id} > w$, $\mathsf{FPFE.ct}_{\mathsf{id},i} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}_{\mathsf{id}}, (i, t_{\mathsf{id},i}, x_{\mathsf{id},i}^{(0)}, 0^{\ell_{\mathcal{X}}}, 0^{\ell_{\mathsf{One\text{-}sFE.ct}}}))$.
   (e) If $i = 1$, set $\mathsf{CT}_{\mathsf{id},1} = (\mathsf{FE.ct}_{\mathsf{id}}, \mathsf{FPFE.ct}_{\mathsf{id},1})$. Else, set $\mathsf{CT}_{\mathsf{id},i} = \mathsf{FPFE.ct}_{\mathsf{id},i}$.
   (f) Send $\mathsf{CT}_{\mathsf{id},i}$ to the adversary.

Lemma 5.20. *If* FPFE *is Q-bounded, function-private, function-selective-IND-secure, then for all PPT adversaries $\mathcal{A}$ and all $w \in [\mathsf{Bound}_{\mathcal{A}}]$,*

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,5}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,6}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

Proof. Suppose for sake of contradiction that there exists a PPT adversary $\mathcal{A}$ and $w \in [\mathsf{Bound}_{\mathcal{A}}]$,

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,5}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,6}(1^\lambda) = 1]\right| > \mathsf{negl}(\lambda) \tag{14}$$

We build a PPT adversary $\mathcal{B}$ that breaks the Q-bounded, function-private, function-selective-IND-security of FPFE. $\mathcal{B}$ first runs steps 1-3 of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,6}$ except that $\mathcal{B}$ does not compute $\mathsf{FPFE.msk}_w$.

For $j \in [Q]$, $\mathcal{B}$ does the following: $\mathcal{B}$ computes $(s_j, \mathsf{One\text{-}sFE.msk}_{j,w}, \mathsf{One\text{-}sFE.Enc.st}_{j,w}, \mathsf{PRF2}.K_{j,w})$ as in step 4 of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,6}$. $\mathcal{B}$ sets $H^{(0)}_{j,w} = H[\mathsf{One\text{-}sFE.msk}_{j,w}, \mathsf{One\text{-}sFE.Enc.st}_{j,w}, \mathsf{PRF2}.K_{j,w}]$ and $H^{(1)}_{j,w} = H'[\mathsf{One\text{-}sFE.msk}_{j,w}, \mathsf{One\text{-}sFE.Enc.st}_{j,w}, \mathsf{PRF2}.K_{j,w}]$.

- $\mathcal{B}$ sets its j^{th} challenge function pair to $(H^{(1)}_{j,w}, H^{(0)}_{j,w})$.

$\mathcal{B}$ then sends all Q challenge function pairs to its FPFE challenger and receives $\{\mathsf{FPFE.sk}_{H_{j,w}}\}_{j\in[Q]}$.

- For each function query f_j output by $\mathcal{A}$, $\mathcal{B}$ computes SK_{f_j} as in step 5a of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,6}$, and sends SK_{f_j} to $\mathcal{A}$.

- For each message query $(x^{(0)}_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i})$ output by $\mathcal{A}$:

  If $\mathsf{id} \neq w$, $\mathcal{B}$ computes $\mathsf{CT}_{\mathsf{id},i}$ as in step 5a of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,6}$, and sends $\mathsf{CT}_{\mathsf{id},i}$ to $\mathcal{A}$.

  If $\mathsf{id} = w$, $\mathcal{B}$ computes $(t_{w,i}, v_{Q,w,i})$ as in step 5b of $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,5}$. $\mathcal{B}$ sends challenge message pair $((i, t_{w,i}, x^{(0)}_{w,i}, x^{(1)}_{w,i}, v_{k,w,i}), (i, t_{w,i}, x^{(1)}_{w,i}, 0^{\ell_{\mathcal{X}}}, 0^{\ell_{\mathsf{FPFE.ct}}}))$ to its FPFE challenger and receives $\mathsf{FPFE.ct}_{w,i}$. This is a valid message query since for all j,

$$H^{(1)}_{j,w}(i, t_{w,i}, x^{(0)}_{w,i}, x^{(1)}_{w,i}, v_{Q,w,i}) = H^{(0)}_{j,w}(i, t_{w,i}, x^{(1)}_{w,i}, 0^{\ell_{\mathcal{X}}}, 0^{\ell_{\mathsf{FPFE.ct}}})$$

  If $i = 1$, $\mathcal{B}$ sets $\mathsf{CT}_{w,1} = (\mathsf{FE.ct}_w, \mathsf{FPFE.ct}_{w,1})$. Else, $\mathcal{B}$ sets $\mathsf{CT}_{w,i} = \mathsf{FPFE.ct}_{w,i}$. $\mathcal{B}$ sends $\mathsf{CT}_{w,i}$ to $\mathcal{A}$.

After $\mathcal{A}$ is done making queries, $\mathcal{A}$ outputs b' which $\mathcal{B}$ also outputs. If the experiment for $\mathcal{A}$ aborts for any reason, $\mathcal{B}$ instead outputs 0. 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 $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,5}$, 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 $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,6}$. Additionally, $\mathcal{B}$ does not need to know $\mathsf{FPFE.msk}_w$ to carry out this experiment and makes only Q function queries. Thus, by Equation 14, this means that $\mathcal{B}$ breaks the Q-bounded, function-private, function-selective-IND-security of FPFE as $\mathcal{B}$ can distinguish between the two security games with non-negligible probability. $\square$

$\mathbf{Hybrid}^{\mathcal{A}}_{1,w,7}(1^\lambda)$: We revert back to using $\mathsf{PRF}.K_w$ to compute the randomness needed for One-sFE, PRF2, and FPFE.KeyGen on stream identity w.

This is the same as $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,6}$ except that we change the following steps:

4 **Precompute Values**: For $j \in [Q]$,

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

(b) $(r_{\mathsf{Setup},j,w}, r_{\mathsf{KeyGen},j,w}, r_{\mathsf{EncSetup},j,w}, r_{\mathsf{PRF2},j,w}, r_{H,j,w}) \leftarrow \mathsf{PRF.Eval}(\mathsf{PRF}.K_w, s_j)$.

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

(d) $\mathsf{One\text{-}sFE.Enc.st}_{j,w} \leftarrow \mathsf{One\text{-}sFE.EncSetup}(\mathsf{One\text{-}sFE.msk}_{j,w}; r_{\mathsf{EncSetup},j,w})$.

(e) $\mathsf{PRF2}.K_{j,w} \leftarrow \mathsf{PRF2.Setup}(1^\lambda; r_{\mathsf{PRF2},j,w})$.

(f) Let $H_{j,w} = H[\mathsf{One\text{-}sFE.msk}_{j,w}, \mathsf{One\text{-}sFE.Enc.st}_{j,w}, \mathsf{PRF2}.K_{j,w}]$ as defined in Figure 10 (page 50).

(g) $\mathsf{FPFE.sk}_{H_{j,w}} = \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}_w, H_{j,w}; r_{H,j,w})$.

5a. **Function Query:** 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) $\mathsf{One\text{-}sFE.sk}_{f_j,w} \leftarrow \mathsf{One\text{-}sFE.KeyGen}(\mathsf{One\text{-}sFE.msk}_{j,w}, f_j; r_{\mathsf{KeyGen},j,w})$.

(b) $c_{j,w} \leftarrow \mathsf{SKE.Enc}(\mathsf{SKE.sk}, (\mathsf{One\text{-}sFE.sk}_{f_j,w}, \mathsf{FPFE.sk}_{H_{j,w}}))$.

(c) Let $G_j = G[f_j, s_j, c_{j,w}]$ as defined in Figure 9 (page 50).

(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.

Lemma 5.21. *If* PRF *is a secure PRF, then for all PPT adversaries* $\mathcal{A}$ *and all* $w \in [\mathsf{Bound}_{\mathcal{A}}]$,

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

Proof. This proof is essentially the same as the proof of Lemma 5.12. $\square$

$\mathbf{Hybrid}^{\mathcal{A}}_{1,w,8}(1^\lambda)$: We change the message encrypted in $\mathsf{FE.ct}_w$ back to its original value.

This is the same as $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,7}$ except that we change the following steps:

3. **Encryption Setup:** For $\mathsf{id} \in [\mathsf{Bound}_{\mathcal{A}}]$ where $\mathsf{Bound}_{\mathcal{A}}$ is a bound on the runtime of $\mathcal{A}$.

   (a) $\mathsf{PRF}.K_{\mathsf{id}} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.

   (b) $\mathsf{FPFE.msk}_{\mathsf{id}} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$.

   (c) ~~If $\mathsf{id} \neq w$,~~ $\mathsf{FE.ct}_{\mathsf{id}} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.ek}, (\mathsf{FPFE.msk}_{\mathsf{id}}, \mathsf{PRF}.K_{\mathsf{id}}, 0, 0^{\ell_{\mathsf{SKE.sk}}}))$.

   (d) ~~If $\mathsf{id} = w$, $\mathsf{FE.ct}_w \leftarrow \mathsf{FE.Enc}(\mathsf{FE.ek}, (0^{\ell_{\mathsf{FPFE.msk}}}, 0^{\ell_{\mathsf{PRF}.K}}, 1, \mathsf{SKE.sk}))$.~~

Lemma 5.22. *If* FE *is a public-key (resp. secret-key) Q-bounded, selective-IND-secure scheme, then for all PPT adversaries $\mathcal{A}$ and all $w \in [\mathsf{Bound}_{\mathcal{A}}]$, for the public-key (resp. secret-key) version of the hybrids,*

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,7}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,8}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

Proof. This proof is essentially the same as the proof of Lemma 5.11. $\square$

$\mathbf{Hybrid}^{\mathcal{A}}_{1,w,9}(1^\lambda)$: For each j, we replace $c_{j,w}$ with a uniform random value c_j. Note that this hybrid is the same as $\mathbf{Hybrid}^{\mathcal{A}}_{1,w+1,0}$.

This is the same as $\mathbf{Hybrid}^{\mathcal{A}}_{1,w,8}$ except that we change the following steps:

2. **Setup:**
   - *Secret-Key Setting:*
     (a) ~~$\mathsf{SKE.sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.~~
     (b) $\mathsf{FE.msk} \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
     (c) $\mathsf{FE.ek} = \mathsf{FE.msk}$.
   - *Public-Key Setting:*
     (a) ~~$\mathsf{SKE.sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.~~
     (b) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
     (c) $\mathsf{FE.ek} = \mathsf{FE.mpk}$.
     (d) Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.

4. **Precompute Values**: Do nothing.

5a. **Function Query:** 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) $c_j \leftarrow \{0,1\}^\lambda$.
   (c) Let $G_j = G[f_j, s_j, c_j]$ as defined in Figure 9 (page 50).
   (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.

Lemma 5.23. *If* SKE *has pseudorandom ciphertexts, then for all PPT adversaries* $\mathcal{A}$ *and all* $w \in [\mathsf{Bound}_{\mathcal{A}}]$,

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,8}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,9}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

Proof. This proof is essentially the same as the proof of Lemma 5.10. □

Lemma 5.24. *For all adversaries* $\mathcal{A}$ *and all* $w \in [\mathsf{Bound}_{\mathcal{A}} - 1]$,

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w,8}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,w+1,0}(1^\lambda) = 1]\right| = 0$$

Proof. The hybrids are identical. □

$\mathbf{Hybrid}_2^{\mathcal{A}}(1^\lambda)$: This is the real world experiment with $b = 1$. This is identical to $\mathbf{Hybrid}^{\mathcal{A}}_{1,\mathsf{Bound}_{\mathcal{A}},9}$.

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:**
   - *Secret-Key Setting:*
     (a) $\mathsf{FE.msk} \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
     (b) $\mathsf{FE.ek} = \mathsf{FE.msk}$.
   - *Public-Key Setting:*
     (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
     (b) $\mathsf{FE.ek} = \mathsf{FE.mpk}$.
     (c) Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.

3. **Encryption Setup:** For $\mathsf{id} \in [\mathsf{Bound}_{\mathcal{A}}]$ where $\mathsf{Bound}_{\mathcal{A}}$ is a bound on the runtime of $\mathcal{A}$.
   (a) $\mathsf{PRF}.K_{\mathsf{id}} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
   (b) $\mathsf{FPFE.msk}_{\mathsf{id}} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$.
   (c) $\mathsf{FE.ct}_{\mathsf{id}} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.ek}, (\mathsf{FPFE.msk}_{\mathsf{id}}, \mathsf{PRF}.K_{\mathsf{id}}, 0, 0^{\ell_{\mathsf{SKE.sk}}}))$.

4. **Precompute Values:** Do nothing.

5. The adversary can make up to Q function queries followed by any polynomial number of message queries.
   (a) **Function Query:** 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:
      i. $s_j \leftarrow \{0,1\}^\lambda$.
      ii. $c_j \leftarrow \{0,1\}^{\ell_{\mathsf{SKE.ct}}}$.
      iii. Let $G_j = G[f_j, s_j, c_j]$ as defined in Figure 9 (page 50).
      iv. $\mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_j)$.
      v. Send $\mathsf{SK}_{f_j} = \mathsf{FE.sk}_{G_j}$ to the adversary.
   (b) **Message Query:** For the i^{th} message query made to stream identity id, $\mathcal{A}$ outputs a message pair $(x^{(0)}_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i})$ where $x^{(0)}_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i} \in \{0,1\}^{\ell_{\mathcal{X}}}$.
      i. $t_{\mathsf{id},i} \leftarrow \{0,1\}^\lambda$.
      ii. $\mathsf{FPFE.ct}_{\mathsf{id},i} \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}_{\mathsf{id}}, (i, t_{\mathsf{id},i}, x^{(1)}_{\mathsf{id},i}, 0^{\ell_{\mathcal{X}}}, 0^{\ell_{\mathsf{One\text{-}sFE.ct}}}))$.
      iii. If $i = 1$, set $\mathsf{CT}_{\mathsf{id},1} = (\mathsf{FE.ct}_{\mathsf{id}}, \mathsf{FPFE.ct}_{\mathsf{id},1})$. Else, set $\mathsf{CT}_{\mathsf{id},i} = \mathsf{FPFE.ct}_{\mathsf{id},i}$.
      iv. Send $\mathsf{CT}_{\mathsf{id},i}$ to the adversary.

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

Lemma 5.25. *For all adversaries $\mathcal{A}$,*

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{1,\mathsf{Bound}_{\mathcal{A}},9}(1^\lambda) = 1] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_2(1^\lambda) = 1]\right| = 0$$

Proof. The hybrids are identical. □

Thus, our lemmas give us the following corollary:

Corollary 5.26. *If*

- One-sFE *is a single-key, single-ciphertext, function-selective-IND-secure, secret-key* sFE *scheme for* P/Poly*,*
- PRF *and* PRF2 *are secure pseudorandom function families,*
- SKE *is a secure secret-key encryption scheme with pseudorandom ciphertexts,*
- FE *is a Q-bounded, selective-IND-secure, public-key (resp. secret-key) FE scheme for* P/Poly*,*
- *and* FPFE *is a Q-bounded, function-private, function-selective-IND-secure, secret-key FE scheme for* P/Poly*,*

then sFE *is a Q-bounded, function-selective-IND-secure, public-key (resp. secret-key) sFE scheme for* P/Poly*.*

Proof. The corollary immediately follows from Lemmas 5.9-5.25. □

Corollary 5.26 then implies Theorem 5.1, since as shown earlier, we can instantiate the required primitives from a Q-bounded, adaptive-IND-secure, public-key (resp. secret-key) FE scheme for P/Poly and a single-key, single-ciphertext, function-selective-IND-secure, secret-key, sFE scheme for P/Poly.

6 Acknowledgements

We thank Manoj Prabhakaran for valuable discussions. This research was supported in part from a Simons Investigator Award, DARPA SIEVE award, NTT Research, NSF grant 2333935, BSF grant 2022370, a Xerox Faculty Research Award, a Google Faculty Research Award, an Okawa Foundation Research Grant, and the Symantec Chair of Computer Science. This material is based upon work supported by the Defense Advanced Research Projects Agency through Award HR00112020024. Part of the research was done while the first author visited University of California, Los Angeles while being supported by IITB Trust Lab. This research was supported in part by the Prime Minister's Research Fellowship, India.

7 References

[ACF+19] Shweta Agrawal, Michael Clear, Ophir Frieder, Sanjam Garg, Adam O'Neill, and Justin Thaler. Ad hoc multi-input functional encryption. Cryptology ePrint Archive, Paper 2019/356, 2019. `https://eprint.iacr.org/2019/356`.

[Agr17] Shweta Agrawal. Stronger security for reusable garbled circuits, general definitions and attacks. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 3–35. Springer, Cham, August 2017.

[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, Berlin, Heidelberg, August 2015.

[AJS15] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Achieving compactness generically: Indistinguishability obfuscation from non-compact functional encryption. *IACR Cryptol. ePrint Arch.*, page 730, 2015.

[AKM+22] Shweta Agrawal, Fuyuki Kitagawa, Anuja Modi, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Bounded functional encryption for Turing machines: Adaptive security from general assumptions. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part I*, volume 13747 of *LNCS*, pages 618–647. Springer, Cham, November 2022.

[AMVY21] Shweta Agrawal, Monosij Maitra, Narasimha Sai Vempati, and Shota Yamada. Functional encryption for Turing machines with dynamic bounded collusion from LWE. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 239–269, Virtual Event, August 2021. Springer, Cham.

[AR17] Shweta Agrawal and Alon Rosen. Functional encryption for bounded collusions, revisited. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 173–205. Springer, Cham, November 2017.

[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, Berlin, Heidelberg, January 2016.

[AV19] Prabhanjan Ananth and Vinod Vaikuntanathan. Optimal bounded-collusion secure functional encryption. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 174–198. Springer, Cham, December 2019.

[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, Cham, November 2017.

[BKS16] Zvika Brakerski, Ilan Komargodski, and Gil Segev. Multi-input functional encryption in the private-key setting: Stronger security from weaker assumptions. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 852–880. Springer, Berlin, Heidelberg, May 2016.

[BPR15] Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a Nash equilibrium. In Venkatesan Guruswami, editor, *56th FOCS*, pages 1480–1498. IEEE Computer Society Press, October 2015.

[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, Berlin, 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.

[CHH$^+$07] Ronald Cramer, Goichiro Hanaoka, Dennis Hofheinz, Hideki Imai, Eike Kiltz, Rafael Pass, Abhi Shelat, and Vinod Vaikuntanathan. Bounded cca2-secure encryption. In *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2007.

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 468–497. Springer, Berlin, Heidelberg, March 2015.

[DGKS24] Pratish Datta, Jiaxin Guan, Alexis Korb, and Amit Sahai. Adaptively secure streaming functional encryption. *IACR Cryptology ePrint Archive*, 2024.

[DKXY02] Yevgeniy Dodis, Jonathan Katz, Shouhuai Xu, and Moti Yung. Key-insulated public key cryptosystems. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 65–82. Springer, Berlin, Heidelberg, April / May 2002.

[GGG$^+$14a] 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, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 578–602. Springer, 2014.

[GGG$^+$14b] 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, Berlin, 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.

[GGL24] Rachit Garg, Rishab Goyal, and George Lu. Dynamic collusion functional encryption and multi-authority attribute-based encryption. In Qiang Tang and Vanessa Teague, editors, *PKC 2024, Part II*, volume 14604 of *LNCS*, pages 69–104. Springer, Cham, April 2024.

[GGLW22] Rachit Garg, Rishab Goyal, George Lu, and Brent Waters. Dynamic collusion bounded functional encryption from identity-based encryption. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 736–763. Springer, Cham, May / June 2022.

[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, Cham, November 2017.

[GJKS15] Vipul Goyal, Abhishek Jain, Venkata Koppula, and Amit Sahai. Functional encryption for randomized functionalities. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 325–351. Springer, Berlin, Heidelberg, March 2015.

[GJO16] Vipul Goyal, Aayush Jain, and Adam O'Neill. Multi-input functional encryption with unbounded-message security. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 531–556. Springer, Berlin, Heidelberg, December 2016.

[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.

[GKS23] Jiaxin Guan, Alexis Korb, and Amit Sahai. Streaming functional encryption. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 433–463. Springer, Cham, August 2023.

[GKW18] Rishab Goyal, Venkata Koppula, and Brent Waters. Collusion resistant traitor tracing from learning with errors. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *50th ACM STOC*, pages 660–670. ACM Press, June 2018.

[GLW12] Shafi Goldwasser, Allison B. Lewko, and David A. Wilson. Bounded-collusion IBE from key homomorphism. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 564–581. Springer, Berlin, Heidelberg, March 2012.

[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.

[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, Cham, April / May 2017.

[GSW21] Rishab Goyal, Ridwan Syed, and Brent Waters. Bounded collusion ABE for TMs from IBE. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 371–402. Springer, Cham, December 2021.

[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, Berlin, Heidelberg, August 2012.

[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, Berlin, Heidelberg, August 2016.

[HK04] Swee-Huay Heng and Kaoru Kurosawa. k-resilient identity-based encryption in the standard model. In Tatsuaki Okamoto, editor, *CT-RSA 2004*, volume 2964 of *LNCS*, pages 67–80. Springer, Berlin, Heidelberg, February 2004.

[ISV+17] Gene Itkis, Emily Shen, Mayank Varia, David Wilson, and Arkady Yerukhimovich. Bounded-collusion attribute-based encryption from minimal assumptions. In Serge Fehr, editor, *PKC 2017, Part II*, volume 10175 of *LNCS*, pages 67–87. Springer, Berlin, Heidelberg, March 2017.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd ACM STOC*, pages 60–73. ACM Press, June 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, Cham, May / June 2022.

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

[SS10] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 2010*, pages 463–472. ACM Press, October 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, Berlin, 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.

[Wee21] Hoeteck Wee. ABE for DFA from LWE against bounded collusions, revisited. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part II*, volume 13043 of *LNCS*, pages 288–309. Springer, Cham, November 2021.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

A Preliminaries Continued

A.1 Standard Notions

Definition A.1 (Pseudorandom Function (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*

- $\mathsf{PRF.Setup}(1^\lambda, 1^n, 1^m)$ *is a randomized algorithm that takes as input the security parameter λ, an input length n, and an output length m, and outputs a key $K \in \mathcal{K}_{\lambda,n,m}$*
- $\mathsf{PRF.Eval}(K, x)$ *is a deterministic algorithm that 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 all PPT *adversaries $\mathcal{A}$,*

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

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

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

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 $K \leftarrow \mathsf{PRF.Setup}(1^\lambda, 1^n, 1^m)$.*
   (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 any polynomial number of times:*
   (a) *$\mathcal{A}$ outputs a value $x \in \{0,1\}^n$.*
   (b) *If $b = 0$, send $y = \mathsf{PRF.Eval}(K, x)$ 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 A.2 (Secret Key Encryption (SKE)). *A secret key encryption scheme with key space $\mathcal{K} = \{\mathcal{K}_\lambda\}_\lambda$ and ciphertext size $m(\cdot)$ is a tuple of PPT algorithms $\mathsf{SKE} = (\mathsf{SKE.Setup}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ where*

- $\mathsf{SKE.Setup}(1^\lambda)$ *is a randomized algorithm that takes as input the security parameter λ and outputs a secret key $\mathsf{sk} \in \mathcal{K}_\lambda$*

- $\mathsf{SKE.Enc}(\mathsf{sk}, x)$ *is a randomized algorithm that takes as input a secret key* $\mathsf{sk} \in \mathcal{K}_{\lambda,n}$ *and a message* $x \in \{0,1\}^*$ *and outputs an encryption* $\mathsf{ct} \in \{0,1\}^{m(\lambda,|x|)}$ *of* x.
- $\mathsf{SKE.Dec}(\mathsf{sk}, \mathsf{ct})$ *is a deterministic algorithm that takes as input a secret key* $\mathsf{sk} \in \mathcal{K}_\lambda$ *and a ciphertext* $\mathsf{ct} \in \{0,1\}^{m(\lambda,n)}$ *for some* n *and outputs a value* $y \in \{0,1\}^n$.

Correctness requires that for all $\lambda, n \in \mathbb{N}$ *and every* $x \in \{0,1\}^n$,

$$\Pr\left[\mathsf{SKE.Dec}(\mathsf{sk}, \mathsf{SKE.Enc}(\mathsf{sk}, x)) = x : \mathsf{sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)\right] = 1$$

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

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

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

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

1. ***Parameters****:* $\mathcal{A}$ *takes as input* 1^λ.
2. ***Setup****:* $\mathsf{sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.
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* $|x_0| = |x_1|$.
   (b) $\mathsf{ct}_b \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}, x_b)$
   (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 secret key encryption scheme has pseudorandom ciphertexts. Intuitively, this means that ciphertexts should be indistinguishable from random strings of the same size.

Definition A.3 (Secret Key Encryption with Pseudorandom Ciphertexts)**.** *A secret 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* $\mathcal{A}$,

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

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

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

1. ***Parameters****:* $\mathcal{A}$ *takes as input* 1^λ.
2. ***Setup****:* $\mathsf{sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$
3. ***Challenge Message Queries****: The following can be repeated any polynomial number of times:*
   (a) $\mathcal{A}$ *outputs a challenge message* x *where* $x \in \{0,1\}^*$.

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

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

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

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

A.2 Secret-Key Functional Encryption

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

Definition A.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:*[21]

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

- $\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct})$*: 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* $1^\lambda \in \mathbb{N}$*, all* $\ell_\mathcal{F}, \ell_\mathcal{X}, \ell_\mathcal{Y} \leq p(1^\lambda)$*, all* $x \in \{0,1\}^{\ell_\mathcal{X}}$*, and all* $f \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$,

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

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

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

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

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

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. *For a polynomial number of rounds, the adversary can do either one of the following in each round:*

   (a) ***Function Query**: The adversary can make at most $Q = Q(\lambda)$ such queries:*

[21] 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.

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

ii. $\mathsf{sk}_f \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}, f)$

iii. *Send* sk_f *to* $\mathcal{A}$

(b) ***Message Query:***

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

ii. $\mathsf{ct} \leftarrow \mathsf{FE.Enc}(\mathsf{msk}, x_b)$.

iii. *Send* ct *to* $\mathcal{A}$.

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

Additionally, when running the experiment, we immediately halt and output 0 *if the adversary ever aborts or if it at any point* $f(x_0) \neq f(x_1)$ *for some message query* (x_0, x_1) *and function query* f *submitted by the adversary.*

Definition A.6 (Other Secret-Key FE Security Definitions). *There are many variations of the security definition. We list a few below:*

- *Q-**Bounded, Selective-IND-Security**: The adversary is required to make all message queries at the beginning of the experiment. This is identical to Definition A.5, except that we do not allow the adversary to make a* Challenge Message Query *after it has made a* Function Query.

- *Q-**Bounded, Function-Selective-IND-Security:*** *The adversary is required to make all function queries at the beginning of the experiment. This is identical to Definition A.5, except that we do not allow the adversary to make a* Function Query *after it has made a* Challenge Message Query.

In the secret-key setting, we can also achieve function privacy. We define it below for the case of Q-bounded, function-selective-IND-security.

Definition A.7 (Q-Bounded, Function-Private, Function-Selective-IND-Security for Secret-Key FE). *A secret-key functional encryption scheme* FE *for* P/Poly *is* Q*-bounded, function-private, function-selective-IND-secure if there exists a negligible function* μ *such that for all* $\lambda \in \mathbb{N}$ *and every PPT adversary* $\mathcal{A}$,

$$\left|\Pr[\mathsf{SKExpt}_{\mathcal{A}}^{\mathsf{FE}\text{-}Q\text{-}\mathsf{FuncPriv}\text{-}\mathsf{FuncSel}\text{-}\mathsf{IND}}(1^\lambda, 0) = 1] - \Pr[\mathsf{SKExpt}_{\mathcal{A}}^{\mathsf{FE}\text{-}Q\text{-}\mathsf{FuncPriv}\text{-}\mathsf{FuncSel}\text{-}\mathsf{IND}}(1^\lambda, 1) = 1]\right| \leq \mu(\lambda)$$

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

$\mathsf{SKExpt}_{\mathcal{A}}^{\mathsf{FE}\text{-}Q\text{-}\mathsf{FuncPriv}\text{-}\mathsf{FuncSel}\text{-}\mathsf{IND}}(1^\lambda, b)$

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 at most* $Q = Q(\lambda)$ *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* $\mathcal{A}$

4. ***Message Queries**: The following can be repeated any polynomial number of times:*
   (a) *$\mathcal{A}$ outputs a message pair (x_0, x_1) where $x_0, x_1 \in \{0,1\}^{\ell_\mathcal{X}}$.*
   (b) $\mathsf{ct} \leftarrow \mathsf{FE.Enc}(\mathsf{msk}, x_b)$.
   (c) *Send* ct *to $\mathcal{A}$.*
5. ***Experiment Outcome**: $\mathcal{A}$ outputs a bit b' which is the output of the experiment.*

Additionally, when running the experiment, we immediately halt and output 0 if the adversary ever aborts or if it at any point $f_0(x_0) \neq f_1(x_1)$ for some message query (x_0, x_1) and function query (f_0, f_1) submitted by the adversary.

A.3 Secret-Key Streaming Functional Encryption

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

Definition A.8 (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:*[22]

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 $y_i \in \{0,1\}^{\ell_\mathcal{Y}}$.*

sFE *must be **streaming efficient**, meaning 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* $\mathsf{poly}(\lambda, \ell_\mathcal{F}, \ell_\mathcal{S}, \ell_\mathcal{X}, \ell_\mathcal{Y})$.

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[23]

[22] 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.

[23] As with all streaming functions, we assume that $\mathsf{Dec.st}_1 = \bot$ if not otherwise specified.

- $\overline{\mathsf{Enc}}(\mathsf{msk}, x)$ *outputs* $\mathsf{ct}_x = (\mathsf{ct}_i)_{i \in [n]}$ *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* $i \in [n]$*.*

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

Definition A.9 (Q-Bounded, Adaptive-IND-Security for Secret-Key sFE)**.** *A secret-key streaming FE scheme* sFE *for* P/Poly *is* Q*-bounded, adaptive-IND-secure if there exists a negligible function* μ *such that for all* $\lambda \in \mathbb{N}$ *and all PPT adversaries* $\mathcal{A}$*,*

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

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

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

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

3. *For a polynomial number of rounds, the adversary can do either one of the following in each round:*

   (a) ***Function Query****: The adversary can make at most* $Q = Q(\lambda)$ *such queries:*

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

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

      iii. *Send* sk_f *to* $\mathcal{A}$*.*

   (b) ***Message Query****:*

      i. $\mathcal{A}$ *outputs a stream identity* id*.*

         a. *If this is the first message query with stream identity* id*, sample* $\mathsf{Enc.st}_{\mathsf{id}} \leftarrow \mathsf{sFE.EncSetup}(\mathsf{msk})$ *and initialize* $\mathsf{index}_{\mathsf{id}} = 1$*. Else, increment* $\mathsf{index}_{\mathsf{id}}$ *by* 1*.*

         b. *Set* $i = \mathsf{index}_{\mathsf{id}}$*.*

      ii. $\mathcal{A}$ *outputs a message pair* $(x_{\mathsf{id},i}^{(0)}, x_{\mathsf{id},i}^{(1)})$ *for stream identity* id *where* $x_{\mathsf{id},i}^{(0)}, x_{\mathsf{id},i}^{(1)} \in \{0,1\}^{\ell_{\mathcal{X}}}$*.*

      iii. $\mathsf{ct}_{\mathsf{id},i} \leftarrow \mathsf{sFE.Enc}(\mathsf{msk}, \mathsf{Enc.st}_{\mathsf{id}}, i, x_{\mathsf{id},i}^{(b)})$*.*

      iv. *Send* $\mathsf{ct}_{\mathsf{id},i}$ *to* $\mathcal{A}$*.*

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

Additionally, when running the experiment, we immediately halt and output 0 *if the adversary ever aborts or if it at any point some function query* f *submitted by the adversary yields different outputs on any of the challenge message streams submitted so far (i.e. if* $f(x_{\mathsf{id}}^{(0)}) \neq f(x_{\mathsf{id}}^{(1)})$ *for some function query* f *submitted by the adversary where* $\{(x_{\mathsf{id},i}^{(0)}, x_{\mathsf{id},i}^{(1)})\}_{i \in [t]}$ *are the message queries submitted so far under some stream identity* id*,* $x_{\mathsf{id}}^{(0)} = x_{\mathsf{id},1}^{(0)} \dots x_{\mathsf{id},t}^{(0)}$*, and* $x_{\mathsf{id}}^{(1)} = x_{\mathsf{id},1}^{(1)} \dots x_{\mathsf{id},t}^{(1)}$*).*

Definition A.10 (Other Secret-Key sFE Security Definitions)**.** *There are many variations of the security definition. We list a few below:*

- *Q-**Bounded, Selective-IND-Security**: The adversary is required to make all message queries before any function queries. This is identical to Definition A.9, except that we do not allow the adversary to make a* Challenge Message Query *after it has made a* Function Query.
- *Q-**Bounded, Function-Selective-IND-Security:** The adversary is required to make all function queries before any message queries. This is identical to Definition A.9, except that we do not allow the adversary to make a* Function Query *after it has made a* Challenge Message Query.

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

Definition A.11 (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* $\mathcal{A}$*,*

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

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

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

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. ***Challenge Message Queries**: The following can be repeated any polynomial number of times:*
   (a) *If this is the first challenge message query, sample* $\mathsf{Enc.st} \leftarrow \mathsf{sFE.EncSetup}(\mathsf{msk})$ *and initialize the index $i = 1$. Else, increment the index i by 1.*
   (b) *$\mathcal{A}$ outputs a challenge message $x_i \in \{0,1\}^{\ell_\mathcal{X}}$.*
   (c) $\mathsf{ct}_i \leftarrow \mathsf{sFE.Enc}(\mathsf{msk}, \mathsf{Enc.st}, i, x_i)$.
   (d) *Send* ct_i *to* $\mathcal{A}$*.*
5. ***Experiment Outcome**: $\mathcal{A}$ outputs a bit b which is the output of the experiment.*

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

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 $1^{\ell_\mathcal{Y}}$.* 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. ***Challenge Message Queries:*** *The following can be repeated any polynomial number of times:*

   (a) *If this is the first challenge message query, initialize the index* $i = 1$ *and set* $\mathsf{st}_1 = \perp$. *Else, increment* i *by 1.*

   (b) $\mathcal{A}$ *outputs a message* $x_i \in \{0,1\}^{\ell_{\mathcal{X}}}$.

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

   (d) Sim *receives* y_i *and outputs a ciphertext* ct_i.

   (e) *Send* ct_i *to* $\mathcal{A}$.

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

Remark A.12. In the secret-key setting, single-key, single-ciphertext, function-selective-SIM-security implies single-key, single-ciphertext, function-selective-IND-security.