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 \( P/\text{Pol}\) 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 $x = x_1 \ldots x_n$, 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 $x = x_1 \ldots x_n$ (denoted $f(x)$) is the sequence of values $y = y_1 \ldots y_n$ resulting from iteratively computing $(y_i, st_{i+1}) = f(x_i, st_i)$ starting from $st_1 = \bot$.

![Diagram of streaming function computation](image)

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. Finally, given a function key for $f$ and access to a stream of ciphertexts $ct_1 \ldots ct_n$ encrypting stream $x = x_1 \ldots x_n$, 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 $f(x) = y_1 \ldots y_n$ 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 (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 $x = x_1 \ldots x_n$ must be generated under the same encryption state $Enc(st)$ 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 $iO$ 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 $iO$ 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 $iO$. 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 $iO$ 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 institutions 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 iO, $Q$-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 $\mathbb{P}/\text{Poly}$ for any polynomial $Q = Q(\lambda)$ of the security parameter $\lambda$.

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 iO.

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 iO-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. 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 iO 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. [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 One-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 OWFs, there exists a single-key, single-ciphertext, function-selectively secure, secret-key sFE scheme for P/Poly.

2. We then bootstrap One-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 P/Poly
(b) a single-key, single-ciphertext, function-selectively secure, secret-key sFE scheme for P/Poly

there exists a Q-bounded, semi-adaptive-function-selectively secure, public-key (resp. secret-key) sFE scheme for 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 One-sFE from compact FE, and [DGKS24] construct an adaptively secure variant of One-sFE using iO and injective PRGs. We remark that since compact FE can be used to build iO [BV15], we only know how to build compact FE from the same assumptions needed to build iO. 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 One-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 One-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 One-sFE. As all prior work crucially required either compact FE or iO, 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 $(y_i, st_{i+1}) = f(x_i, st_i)$. 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.\(^\text{10}\)

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 $gf, st_{i+1}$ (where $st_{1} = \bot$ and $gf, st_{1}$ 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 $gf, st_{i}$ 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.

![Figure 2](image-url)

**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 $x = x_1 \ldots x_n$ 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 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 \( g_{f, st_{i+1}} \). 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 \( g_{f, st_{i+1}} \). 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 \( \bar{st}_i \) 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 One-sFE scheme. We will then modify each function \( g_{f, \bar{st}_i} \) (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, \bar{st}_i}(x_i, p_i, p_{i+1}, msk_{i+1}): \\
1. st_i = \bar{st}_i \oplus p_i \\
2. (y_i, st_{i+1}) = f(x_i, st_i). \\
3. st_{i+1} = st_{i+1} \oplus p_{i+1} \\
4. Output \((y_i, FE.KeyGen(msk_{i+1}, g_{f, \bar{st}_{i+1}}))\).
\]

Figure 3: Definition of \( g_{f, \bar{st}_i} \).

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 \( \ell \) denote the length of circuits supported by our FE scheme, and let SKE be a secret-key encryption scheme.

- **Setup(1\(^\lambda\))**: Generate \( 2\ell \) SKE keys \( \{sk_{j,b}\}_{j \in [\ell], b \in \{0,1\}} \). Output these keys as the MSK.

- **Enc(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 \( \tilde{U} \) and input labels \( \{\text{lab}_{j,b}\}_{j \in [\ell], b \in \{0,1\}} \). Encrypt each label \( \text{lab}_{j,b} \) under \( sk_{j,b} \) to get \( ct_{j,b} \). Output \( CT = (\tilde{U}, \{ct_{j,b}\}_{j \in [\ell], b \in \{0,1\}}) \).

- **KeyGen(MSK, g)**: Output \( SK_g = \{sk_{j,g[j]}\}_{j \in [\ell]} \) where \( g[j] \) is the \( j^{th} \) bit of \( g \).

- **Dec(SK_g, CT_x)**: Use the secret keys \( \{sk_{j,g[j]}\}_{j \in [\ell]} \) from \( SK_g \) to decrypt the corresponding ciphertexts from \( CT_m \) and recover \( \{\text{lab}_{j,g[j]}\}_{j \in [\ell]} \). Then, evaluate the garbled circuit \( \tilde{U} \) on these labels to learn \( U_x(g) = g(x) \).

![Figure 4: FE scheme from [SS10].](image)

Correctness follows from the correctness of SKE and the garbling scheme. To argue security, we can first switch all ciphertexts \( ct_{j,1-g[j]} \) for the unused labels \( \text{lab}_{j,1-g[j]} \) to encryptions of \( \perp \) since the corresponding secret keys \( sk_{j,1-g[j]} \) are kept hidden from the adversary. We can then use the security of the garbling scheme to simulate both the garbled circuit \( \tilde{U} \) for \( x \) and the input labels \( \{\text{lab}_{j,g[j]}\}_{j \in [\ell]} \) for \( g \) from \( (g, g(x) = U_x(g)) \). Since the circuit \( \tilde{U} \) 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 universal 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 \( g_{f,\tilde{x}_i} \). Unfortunately, as previously mentioned, we would still have an efficiency/size problem with our function keys since the definition of \( g_{f,\tilde{x}_i} \) 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 $st_i$ to state $st_{i+1}$. 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:

$$U[x_i, p_i, p_{i+1}, msk_{i+1}](g_{f,st_i}):$$

1. Output $g_{f,st_i}(x_i, p_i, p_{i+1}, msk_{i+1})$

where we define $g_{f,st_i}$ 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 $msk_{i+1} = \{sk_{i+1,j,b}\}_{j,b}$ along with [SS10]'s key generation algorithm which simply outputs a selection of SKE keys. We use $(f, st_{i+1})[j]$ to denote the $j^{th}$ bit of the tuple $(f, st_{i+1})$.

$$U[x_i, p_i, p_{i+1}, \{sk_{i+1,j,b}\}_{j,b}](f, st_i):$$

1. $p_i = st_i \oplus p_i$.
2. $(y_i, st_{i+1}) = f(x_i, st_i)$.
3. $st_{i+1} = st_{i+1} \oplus p_{i+1}$.
4. Output $(y_i, \{sk_{i+1,j,(f,st_{i+1})[j]}\}_{j,b})$.

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 $sk_{i+1,j,(f,st_{i+1})[j]}$ 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 $st_{i+1}$, it only outputs the secret keys corresponding to $st_{i+1}$. Since the function $f$ does not change between iterations, we can provide the SKE keys corresponding to $f$ directly in the One-sFE function key for $f$. More precisely, the One-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 $\ell_x$ and $\ell_s$ 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.

- **Setup(1^\lambda):** Generate PRF keys for computing pads $p_i$ along with $2(\ell_x + \ell_s)$ SKE keys $\{sk_{i,j,b}\}_{j,e(\ell_x),b(0,1)}$, $\{sk_{i,k,b}\}_{k(\ell_s),b(0,1)}$ for every iteration $i$. Output these PRF keys as the master secret key MSK.
• EncSetup(MSK): Output Enc.st = ⊥.

• Enc(MSK, Enc.st, i, x_i):
  1. Use MSK to generate the $2(\ell_F + \ell_S)$ SKE keys for indices $i$ and $i + 1$.
  2. Define circuit $U_i = U[x_i, p_i, p_{i+1}, \{sk_{i+1,j,b}\}_{j,b}, \{sk_{i+1,k,b}\}_{k,b}]$ as below where $U_i$ has input $x_i$, the pads $p_i$ and $p_{i+1}$, and the secret keys $\{sk_{i+1,j,b}\}_{j \in [\ell_F]}$, $\{sk_{i+1,k,b}\}_{k \in [\ell_S], b \in \{0,1\}}$ for the next iteration hardwired into it.
  3. Garble $U_i$ using a circuit garbling scheme to get $e[U_i]$ and input labels $\{lab_{i,j,b}\}_{j \in [\ell_F], b \in \{0,1\}}$, $\{lab_{i,k,b}\}_{k \in [\ell_S], b \in \{0,1\}}$ 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 $\tilde{st}_i$.
  4. Encrypt each label $lab_{i,j,b}$ under $sk_{i,j,b}$ to get $ct_{i,j,b}$. Similarly, encrypt each label $lab_{i,k,b}$ under $sk_{i,k,b}$ to get $ct_{i,k,b}$.
  5. Output $CT_i = (e[U_i], \{ct_{i,j,b}\}_{j,b}, \{ct_{i,k,b}\}_{k,b})$.

• KeyGen(MSK, f): Use MSK to compute the SKE keys $\{sk_{i,k,\tilde{st}_1[k]}\}_{k \in [\ell_S]}$ for the first encrypted state $\tilde{st}_1 = p_1 \oplus st_1$. Then, use MSK to generate a limited selection of PRF keys which will allow the user to compute $\{sk_{i,j,f[j]}\}_{j \in \ell_F}$ for every iteration $i$, but no other SKE keys. Output these PRF keys along with the SKE keys for $\tilde{st}_1$ as the function key $SK_f$.

• Dec(SK_f, Dec.st_i, i, CT_i): Use the SKE keys for $f$ and $\tilde{st}_{i+1}$ provided in $SK_f$ and/or $Dec.st_i$ to decrypt the corresponding ciphertexts from $CT_i$ and recover $\{lab_{i,j,f[j]}\}_{j \in \ell_F}$, $\{lab_{i,k,\tilde{st}_i[k]}\}_{k \in [\ell_S]}$. Then, evaluate the garbled circuit $\tilde{U}_i$ from $CT_i$ on these labels to learn $U_i(f, \tilde{st}_i) = (y_i, \{sk_{i+1,k,\tilde{st}_{i+1}[k]}\}_k)$. Output $y_i$ and $Dec.st_{i+1} = \{sk_{i+1,k,\tilde{st}_{i+1}[k]}\}_k$. 

12
1. $p_i = st_i \oplus p_i$.
2. $(y_i, st_{i+1}) = f(x_i, st_i)$.
3. $\tilde{st}_{i+1} = st_{i+1} \oplus p_{i+1}$.
4. Output $(y_i, \{sk_{i+1,k,\tilde{st}_{i+1}}\k, b\})$.

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 $\{ct_{i,j,1-f[j]}\}_j, \{ct_{i,k,1-\tilde{st}[k]}\}_k$ for the unused labels $\{ct_{i,j,1-f[j]}\}_j, \{ct_{i,k,1-\tilde{st}[k]}\}_k$ to encryptions of $\bot$ 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 $\tilde{U}_i$ for $x_i$ and the input labels $\{lab_{i,j,f[j]}\}_j \in [x], \{lab_{i,j,\tilde{st}[j]}\}_j \in [s]$ from $((f, \tilde{st}_i), U_i(f, \tilde{st}_i))$. 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 $\tilde{s}_i$, the intermediate states $s_i$ 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.\footnote{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]).} 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 $iO$) 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 $One\text{-sFE.msk}$ and a corresponding One-sFE function key $One\text{-sFE.sk}_f$ for every (function, stream) pair.

- **FPFE**: an (unbounded-collusion) function-private, secret-key FE scheme.\footnote{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].} 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:\footnote{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 (FE.mpk, 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 $FE.sk_G$ for the function $G_{f,s}$ defined below where $s$ is a random value. $G_{f,s}$ will generate a fresh One-sFE master secret key One-sFE.msk along with a corresponding encryption state One-sFE.Enc.st and a corresponding function key One-sFE.sk$_f$ for $f$. The output of $G_{f,s}$ will be One-sFE.sk$_f$ and an FPFE ciphertext encrypting (1) One-sFE.msk, (2) One-sFE.Enc.st and (3) a PRF key.

To encrypt a stream $x = x_1 \ldots x_n$:

1. We will first create an FE ciphertext $FE.ct$ of (FPFE.msk, PRF.K) where FPFE.msk is an FPFE master secret key and PRF.K is a PRF key. We will provide $FE.ct$ 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 $FPFE.sk_H_i$ 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 One-sFE master secret key (and a few other needed values) and output a One-sFE encryption One-sFE.ct$_i$ of $x_i$.

To decrypt:

1. We can first use FE to decrypt $FE.ct$ (from our first sFE ciphertext) with FE.sk$_G$ (from our sFE function key) to get One-sFE.sk$_f$ and an FPFE ciphertext FPFE.ct.

2. We can then use FPFE to decrypt FPFE.ct with FPFE.sk$_H_i$ (from the $i^{th}$ sFE ciphertext) to get One-sFE.ct$_i$.

3. Finally, we can use One-sFE to decrypt each One-sFE.ct$_i$ using One-sFE.sk$_f$ to learn the corresponding output values $y_i$. 
Figure 6: \([\text{GKS23}]\)'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 \((\text{One-sFE}.\text{sk}_f, \text{FPFE}.\text{ct})\) inside the function key for \(G_{f,s}\) as part of an SKE ciphertext, where \(\text{FPFE}.\text{ct}\) is encrypting the tuple \((\text{One-sFE}.\text{msk}, \text{One-sFE}.\text{Enc}.\text{st}, \text{PRF2}.K)\), 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 $\text{FPFE}.\text{msk}$ and $\text{PRF}.K$ from the FE ciphertext, so that we can now use FPFE security to hardwire the One-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 $(\text{One-sFE}.\text{msk}, \text{One-sFE.Enc.st}, \text{PRF2}.K)$ from the FPFE ciphertext and $x_i$ from the FPFE function key for $H_i$, so that we can now use One-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 One-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 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.
**Figure 7**: Our technique for bootstrapping to public-key sFE.

- **Encrypted Under**: $FE.mpk, FE.msk$
- **Function**: $G_{f,s}$
- **Message**: $FPFE.mpk, PRF.K$

**$G_{f,s}(FPFE.mpk, PRF.K)$:**

1. $(r_{Setup}, r_{EncSetup}, r_{KeyGen}, r_{PRF2}, r_{Enc}) \leftarrow PRF.Eval(PRF.K, s)$.
2. One-sFE.msk $\leftarrow$ One-sFE.Setup($1^\lambda; r_{Setup}$).
3. One-sFE.Enc.st $\leftarrow$ One-sFE.EncSetup(One-sFE.msk; $r_{EncSetup}$).
4. One-sFE.sk_f $\leftarrow$ One-sFE.KeyGen(One-sFE.msk, $f; r_{KeyGen}$).
5. PRF2.K $\leftarrow$ PRF2.Setup($1^\lambda; r_{PRF2}$).
6. FPFE.sk_H $\leftarrow$ FPFE.KeyGen(FPFE.mpk, $H[One-sFE.mpk, One-sFE.Enc.st, PRF2.K]; r_{Enc}$).
7. Output $(One-sFE.sk_f, FPFE.sk_H)$.

**$H[One-sFE.mpk, One-sFE.Enc.st, PRF2.K](i, x_i, t_i)$:**

1. $r_i \leftarrow PRF2.Eval(PRF2.K, t_i)$
2. Output One-sFE.Enc(One-sFE.mpk, One-sFE.Enc.st, $i, x_i; r_i$)
3 Preliminaries

Throughout, we will use $\lambda$ to denote the security parameter.

Notation

- We say that a function $f(\lambda)$ is negligible in $\lambda$ if $f(\lambda) = \lambda^{-\omega(1)}$, and we denote it by $f(\lambda) = \text{negl}(\lambda)$.
- We say that a function $g(\lambda)$ is polynomial in $\lambda$ if $g(\lambda) = p(\lambda)$ for some fixed polynomial $p$, and we denote it by $g(\lambda) = \text{poly}(\lambda)$.
- 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 $\text{GC} = (\text{Garble}, \text{Eval})$ defined as follows:

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

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

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

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

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

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

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

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

1. $A$ takes as input $1^\lambda$ and outputs $(C, x)$ where $C$ is a circuit with $n$-bit inputs and $x \in \{0,1\}^n$.
2. If $i = 0$, $(\tilde{C}, \{\text{lab}_{k,b}\}_{k \in [n], b \in \{0,1\}}) \leftarrow \text{Garble}(1^\lambda, C)$.
3. If $i = 1$, $(\tilde{C}, \{\text{lab}_{k,x[k]}\}_{k \in [n]}) \leftarrow \text{Sim}(1^\lambda, 1^{|C|}, x, C(x))$.
4. Send $(\tilde{C}, \{\text{lab}_{k,x[k]}\}_{k \in [n]})$ to $A$. 
Lemma 3.2 ([Yao86]). If there exist OWFs, 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 $F[\ell_F, \ell_X, \ell_Y]$ is the set of all functions $f$ that have a description $\widetilde{f} \in \{0,1\}^{\ell_F}$, take inputs in $\{0,1\}^{\ell_X}$, and output values in $\{0,1\}^{\ell_Y}$.

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 $FE = (\text{Setup}, \text{Enc}, \text{KeyGen}, \text{Dec})$ defined as follows:\footnote{We also allow $\text{Enc}, \text{KeyGen},$ and $\text{Dec}$ to additionally receive parameters $1^\lambda, 1^{\ell_F}, 1^{\ell_X}, 1^{\ell_Y}$ as input, but omit them from our notation for convenience.}

- $\text{Setup}(1^\lambda, 1^{\ell_F}, 1^{\ell_X}, 1^{\ell_Y})$: takes as input the security parameter $\lambda$, a function size $\ell_F$, an input size $\ell_X$, and an output size $\ell_Y$, and outputs the master public key $\text{mpk}$ and the master secret key $\text{msk}$.

- $\text{Enc}(\text{mpk}, x)$: takes as input the master public key $\text{mpk}$ and a message $x \in \{0,1\}^{\ell_X}$, and outputs an encryption $\text{ct}$ of $x$.

- $\text{KeyGen}(\text{msk}, f)$: takes as input the master secret key $\text{msk}$ and a function $f \in F[\ell_F, \ell_X, \ell_Y]$, and outputs a function key $\text{sk}_f$.

- $\text{Dec}(\text{sk}_f, \text{ct})$: takes as input a function key $\text{sk}_f$ and a ciphertext $\text{ct}$, and outputs a value $y \in \{0,1\}^{\ell_Y}$.

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

$$\Pr\left[\text{Dec}(\text{sk}_f, \text{ct}_x) = f(x) : (\text{mpk}, \text{msk}) \leftarrow \text{Setup}(1^\lambda, 1^{\ell_F}, 1^{\ell_X}, 1^{\ell_Y}), \text{ct}_x \leftarrow \text{Enc}(\text{mpk}, x), \text{sk}_f \leftarrow \text{KeyGen}(\text{msk}, f)\right] \geq 1 - \mu(\lambda).$$

**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 $\mu$ such that for all $\lambda \in \mathbb{N}$ and every PPT adversary $A$,

$$\Pr_{A}[\text{Exp}_A^{FE-Q-\text{Ad-IND}}(1^{\lambda}, 0) = 1] - \Pr_{A}[\text{Exp}_A^{FE-Q-\text{Ad-IND}}(1^{\lambda}, 1) = 1] \leq \mu(\lambda)$$

where for each $b \in \{0,1\}$ and $\lambda \in \mathbb{N}$, we define
Expt_{\mathcal{A}}^{\text{FE-Q-Ad-IND}}(1^\lambda, b)

1. **Parameters**: \( \mathcal{A} \) takes as input \( 1^\lambda \), and outputs a function size \( 1^{\ell_F} \), an input size \( 1^{\ell_X} \), and an output size \( 1^{\ell_Y} \).

2. **Setup**: \((\text{mpk, msk}) \leftarrow \text{FE.Setup}(1^\lambda, 1^{\ell_F}, 1^{\ell_X}, 1^{\ell_Y}).\)

3. **Public Key**: Send \text{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 \( f \in \mathcal{F}[\ell_F, \ell_X, \ell_Y] \)
       - ii. \( \text{sk}_f \leftarrow \text{FE.KeyGen}(\text{msk, } f) \)
       - iii. Send \text{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 \( x_0, x_1 \in \{0, 1\}^{\ell_X} \).
       - ii. \( \text{ct} \leftarrow \text{FE.Enc}(\text{mpk, } x_b) \)
       - iii. Send \text{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 make 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 encryption (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 $S$, input space $X$, and output space $Y$ is a function $f : X \times S \rightarrow Y \times S$.

- We define the output of $f$ on $x = x_1 \ldots x_n \in X^n$ (denoted $f(x)$) to be $y = y_1 \ldots y_n \in Y^n$ where\(^{15}\) we have $st_1 = \perp$ and

\[(y_i, st_{i+1}) = f(x_i, st_i)\]

**Definition 3.12** (Streaming Function Class). The streaming function class $F[\ell_F, \ell_S, \ell_X, \ell_Y]$ is the set of all streaming functions $f$ that have a description $f \in \{0, 1\}^{\ell_F}$, state space $S = \{0, 1\}^{\ell_S}$, input space $X = \{0, 1\}^{\ell_X}$, and output space $Y = \{0, 1\}^{\ell_Y}$.

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 $sFE = (\text{Setup}, \text{EncSetup}, \text{Enc}, \text{KeyGen}, \text{Dec})$ defined as follows:\(^{16}\)

- $\text{Setup}(1^\lambda, 1^{\ell_F}, 1^{\ell_S}, 1^{\ell_X}, 1^{\ell_Y})$: takes as input the security parameter $\lambda$, a function size $\ell_F$, a state size $\ell_S$, an input size $\ell_X$, and an output size $\ell_Y$, and outputs the master public key $\text{mpk}$ and the master secret key $\text{msk}$.

\(^{15}\)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 $\text{Enc}$, $\text{EncSetup}$, $\text{KeyGen}$, and $\text{Dec}$ to additionally receive parameters $1^\lambda, 1^{\ell_F}, 1^{\ell_S}, 1^{\ell_X}, 1^{\ell_Y}$ as input, butomit them from our notation for convenience.
• EncSetup(mpk): takes as input the master public key mpk and outputs an encryption state Enc.st.

• Enc(mpk, Enc.st, i, x_i): takes as input the master public key mpk, an encryption state Enc.st, an index i, and a message x_i ∈ {0, 1}^{ℓ_x} and outputs an encryption ct_i of x_i.

• KeyGen(msk, f): takes as input the master secret key msk, and a function f ∈ F[ℓ_F, ℓ_S, ℓ_X, ℓ_Y] and outputs a function key sk_f.

• Dec(sk_f, Dec.st_i, i, ct_i): where for each function key sk_f, Dec(sk_f, ·, ·) is a streaming function that takes as input a state Dec.st_i, an index i, and an encryption ct_i and outputs a new state Dec.st_{i+1} and an output y_i ∈ {0, 1}^{ℓ_Y}.

sFE must be streaming efficient, meaning that the size and runtime of all algorithms of sFE on security parameter λ, function size ℓ_F, state size ℓ_S, input size ℓ_X, and output size ℓ_Y are poly(λ, ℓ_F, ℓ_S, ℓ_X, ℓ_Y).

sFE satisfies correctness if for all polynomials p, there exists a negligible function µ such that for all λ ∈ N, all ℓ_F, ℓ_S, ℓ_X, ℓ_Y ≤ p(λ), all n ∈ [2^λ], all x = x_1 . . . x_n where each x_i ∈ {0, 1}^{ℓ_x}, and all f ∈ F[ℓ_F, ℓ_S, ℓ_X, ℓ_Y],

\[
\Pr \left[ \text{Dec}(sk_f, ct_x) = f(x) : \begin{align*}
    & (mpk, msk) \leftarrow \text{Setup}(1^λ, 1^ℓ_F 1^ℓ_S, 1^ℓ_X, 1^ℓ_Y), \\
    & ct_x \leftarrow \text{Enc}(mpk, x) \\
    & sk_f \leftarrow \text{KeyGen}(msk, f)
\end{align*} \right] \geq 1 - µ(λ)
\]

where we define^{17}

• Enc(mpk, x) outputs ct_x = (ct_i)_{i ∈ [n]} produced by sampling Enc.st ← EncSetup(mpk) and then computing ct_i ← Enc(mpk, Enc.st, i, x_i) for i ∈ [n].

• Dec(sk_f, ct_i) outputs y_i = (y_i)_{i ∈ [n]} where (y_i, Dec.st_{i+1}) = Dec(sk_f, Dec.st_i, i, ct_i) for i ∈ [n].

Definition 3.14 (Q-Bounded, Adaptive-IND-Security for Public-Key sFE). A public-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 λ ∈ N and all PPT adversaries A,

\[
|\Pr_{\text{A}}[\text{Expt}_{1}^{\text{sFE-Q-Ad-IND}}(1^λ, 0) = 1] - \Pr_{\text{A}}[\text{Expt}_{1}^{\text{sFE-Q-Ad-IND}}(1^λ, 1) = 1]| \leq µ(λ)
\]

where for each b ∈ {0, 1} and λ ∈ N, we define

\[
\text{Expt}_{\text{A}}^{\text{sFE-Q-Ad-IND}}(1^λ, b)
\]

1. Parameters: A takes as input 1^λ, and outputs a function size 1^ℓ_F, a state size 1^ℓ_S, an input size 1^ℓ_X, and an output size 1^ℓ_Y.

2. Setup: (mpk, msk) ← sFE.Setup(1^λ, 1^ℓ_F, 1^ℓ_S, 1^ℓ_X, 1^ℓ_Y).

3. Public Key: Send mpk to 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(λ) such queries:

^{17}As with all streaming functions, we assume that Dec.st_t = ⊥ if not otherwise specified.
i. A outputs a streaming function query \( f \in \mathcal{F}[\ell_F, \ell_S, \ell_X, \ell_Y] \).

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

iii. Send \( \text{sk}_f \) to \( A \).

(b) Challenge Message Query:
   i. If this is the first challenge message query, sample \( \text{Enc.st} \leftarrow \text{sFE.EncSetup}(\text{mpk}) \)
      and initialize the index \( i = 1 \). Else, increment the index \( i \) by 1.
   ii. A outputs a challenge message pair \((x_i^{(0)}, x_i^{(1)})\) where \( x_i^{(0)}, x_i^{(1)} \in \{0,1\}^{\ell_X} \).
   iii. \( \text{ct}_i \leftarrow \text{sFE.Enc}(\text{mpk}, \text{Enc.st}, i, x_i^{(b)}) \).
   iv. Send \( \text{ct}_i \) to \( A \).

5. Experiment Outcome: 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, \( x^{(0)} = x_1^{(0)} \ldots x_t^{(0)} \), and \( x^{(1)} = x_1^{(1)} \ldots x_t^{(1)} \).

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 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 OWFs, 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.**

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

4.1 Parameters

On security parameter \( 1^\lambda \), function size \( \ell_F \), state size \( \ell_S \), input size \( \ell_X \), and output size \( \ell_Y \), we will instantiate our primitives with the following parameters:

- We instantiate our \( \text{PRF}_s \) with the following parameters:

<table>
<thead>
<tr>
<th></th>
<th>Security Parameter</th>
<th>Input Size</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{PRF}_1 )</td>
<td>( \lambda )</td>
<td>( \log(\ell_F) + 1 )</td>
<td>( \lambda )</td>
</tr>
<tr>
<td>( \text{PRF}_2 )</td>
<td>( \lambda )</td>
<td>( \lambda )</td>
<td>( \lambda )</td>
</tr>
<tr>
<td>( \text{PRF}_3 )</td>
<td>( \lambda )</td>
<td>( \lambda + \log(\ell_S) + 1 )</td>
<td>( \lambda )</td>
</tr>
<tr>
<td>( \text{PRF}_p )</td>
<td>( \lambda )</td>
<td>( \lambda )</td>
<td>( \ell_S )</td>
</tr>
</tbody>
</table>

- \( \text{SKE} \): We instantiate \( \text{SKE} \) with security parameter \( \lambda \). We will use \( \text{SKE} \) to encrypt messages of length \( \lambda \).
- \( \text{GC} \): We instantiate \( \text{GC} \) with security parameter \( \lambda \). We will use \( \text{GC} \) to garble circuits of the form \( U[x_i, p_i, p_{i+1}, \{sk'_i+1,k,b\}_{k \in \ell_S, b \in \{0,1\}}] \) as defined in Figure 8 where \( x_i \in \{0,1\}^{\ell_X}, p_i, p_{i+1} \in \{0,1\}^{\ell_S} \), each \( sk'_i+1,k,b \) is a secret key of \( \text{SKE} \), and which takes inputs of size \( \ell_F + \ell_S \). Let \( \ell_U \) be the size of such circuits. Recall that each of the input labels of a garbled circuit are size \( \lambda \).

**Remark 4.2.** We assume without loss of generality that for security parameter \( \lambda \), all algorithms only require randomness of length \( \lambda \). If the original algorithm required additional randomness, we can replace it with a new algorithm that first expands the \( \lambda \) 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 \( \ell_F, \ell_S, \ell_X, \ell_Y \) are polynomial in \( \lambda \)).
4.2 Construction

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

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

- \textbf{One-sFE.Setup}(1\^λ, 1\^ℓ_x, 1\^ℓ_s, 1\^ℓ_y):
  1. \(K \leftarrow \text{PRF_1.Setup}(1\^λ), K' \leftarrow \text{PRF_3.Setup}(1\^λ), K_p \leftarrow \text{PRF_p.Setup}(1\^λ).\)
  
  * Throughout, for \(i \in [2^λ], j \in [\ell_x], k \in [\ell_s], b \in \{0, 1\},\) we will define
    \[
    K_{j,b} = \text{PRF_2.Setup}(1\^λ; \text{PRF_1.Eval}(K, (j, b)))
    \]
    \[
    sk_{i,j,b} = \text{SKE.Setup}(1\^λ; \text{PRF_2.Eval}(K_{j,b}, i))
    \]
    \[
    sk_{i,k,b}' = \text{SKE.Setup}(1\^λ; \text{PRF_3.Eval}(K', (i, k, b)))
    \]
    \[
    p_i = \text{PRF_p.Eval}(K_p, i)
    \]

  Observe that these values can be computed from \(i, j, k, b, K, K',\) and \(K_p.\)

  2. Output MSK = \((K, K', K_p).\)

- \textbf{One-sFE.EncSetup}(MSK): Output \text{Enc.st} = ⊥.

- \textbf{One-sFE.Enc}(MSK, Enc.st, \(i, x_i):\)
  1. Parse MSK = \((K, K', K_p).\)

  2. Garble circuit:
    - (a) Compute \(p_i, p_{i+1}, \{sk_{i+1,k,b}'\}_{k \in [\ell_s], b \in \{0, 1\}}\) from \(K', K_p.\)
    - (b) Let \(U_i = U[x_i, p_i, p_{i+1}, \{sk_{i+1,k,b}'\}_{k \in [\ell_s], b \in \{0, 1\}}]\) as defined in Figure 8.
    - (c) \((\bar{U}_i, \{\text{lab}_{i,j,b}\}_{j \in [\ell_x], b \in \{0,1\}}, \{\text{lab}_{i,k,b}'\}_{k \in [\ell_s], b \in \{0,1\}}) \leftarrow \text{GC.Garble}(1\^λ, U_i).^{18}\)

  3. Encrypt function labels: For \(j \in [\ell_x], b \in \{0, 1\},\)
    - (a) Compute \(sk_{i,j,b}\) from \(K.\)
    - (b) \(ct_{i,j,b} \leftarrow \text{SKE.Enc}(sk_{i,j,b}, \text{lab}_{i,j,b}).\)

  4. Encrypt state labels: For \(k \in [\ell_s], b \in \{0, 1\},\)
    - (a) Compute \(sk_{i,k,b}'\) from \(K'.\)
    - (b) \(ct_{i,k,b}' \leftarrow \text{SKE.Enc}(sk_{i,k,b}', \text{lab}_{i,k,b}').\)

  5. Output \(CT_i = (\bar{U}_i, \{ct_{i,j,b}\}_{j \in [\ell_x], b \in \{0,1\}}, \{ct_{i,k,b}'\}_{k \in [\ell_s], b \in \{0,1\}}).\)

\footnote{\textsuperscript{18}For notational convenience, we split the input labels for \(\bar{U}_i\) into two categories depending upon what part of the input they represent. We use \(\{\text{lab}_{i,j,b}\}_{j \in [\ell_x], b \in \{0,1\}}\) to refer to the labels for the part of the input representing \(f\) (i.e. the first \(\ell_x\) bits of the input), and use \(\{\text{lab}_{i,k,b}'\}_{k \in [\ell_s], b \in \{0,1\}}\) to refer to the labels for the part of the input representing \(\text{st}_i\) (i.e. the last \(\ell_s\) bits of the input).}
\[ U[x_i, p_i, p_i+1, \{sk_i^{i+1, k, b}\}_{k \in \ell, b \in \{0, 1\}}](f, \tilde{s}_t): \]

1. \( st_i = st_i \oplus p_i \).
2. \((y_i, st_{i+1}) = f(x_i, st_i)\).
3. \( \tilde{s}_{t+1} = st_{i+1} \oplus p_{i+1} \).
4. Output \((y_i, (\tilde{s}_{t+1}, \{sk_i^{i+1, k, \tilde{s}_{t+1} + 1}\}_{k \in \ell, b \in \{0, 1\}})\).

Figure 8: Definition of \( U[x_i, p_i, p_i+1, \{sk_i^{i+1, k, b}\}_{k \in \ell, b \in \{0, 1\}}] \)

- **One-sFE.KeyGen**(MSK, \( f \))
  1. Parse MSK = \((K, K', K_p)\).
  2. Compute PRF keys for generating SKE keys for \( f \):
     a. Compute \( \{K_j, f[j]\}_{j \in \ell} \) from \( K \).
  3. Compute SKE keys for \( \tilde{s}_t \):
     a. Compute \( p_1 \) from \( K_p \).
     b. \( \tilde{s}_t = p_1 \).
     (Here, we assume \( s_t = 0^k \) for all streaming functions so that \( \tilde{s}_t = s_t \oplus p_1 = p_1 \).)
     c. Compute \( \{sk_i^{i+1, \tilde{s}_t, k}\}_{k \in \ell} \) from \( K' \).
  4. Output \( SK_f = (f, \{K_j, f[j]\}_{j \in \ell}, (\tilde{s}_t, \{sk_i^{i+1, \tilde{s}_t, k}\}_{k \in \ell, b \in \{0, 1\}}) \).

- **One-sFE.Dec**(SK\(_f\), Dec.st\(_t\), i, CT\(_i\)):
  1. Parse SK\(_f\) = \((f, \{K_j, f[j]\}_{j \in \ell}, (\tilde{s}_t, \{sk_i^{i+1, \tilde{s}_t, k}\}_{k \in \ell, b \in \{0, 1\}})\)).
  2. If \( i > 1 \), parse Dec.st\(_t\) = \((\tilde{s}_t, \{sk_i^{i+1, \tilde{s}_t, k}\}_{k \in \ell})\).
  3. Parse CT\(_i\) = \((\tilde{U}_i, \{ct_i, j, b\}_{j \in \ell, b \in \{0, 1\}, \{ct_i^{i+1, k,b}\}_{k \in \ell, b \in \{0, 1\}}, \{sk_i^{i+1, k, \tilde{s}_t, k}\}_{k \in \ell, b \in \{0, 1\}})\).
  4. Recover function labels: For \( j \in \ell \),
     a. \( r_{i,j,f[j]} = PRF_2.Eval(K_j, f[j], i) \).
     b. \( sk_{i,j,f[j]} = SKE.Setup(1_\lambda; r_{i,j,f[j]}) \).
     c. \( lab_{i,j,f[j]} = SKE.Dec(sk_{i,j,f[j]}, ct_{i,j,f[j]}) \).
  5. Recover state labels: For \( k \in \ell \),
     a. \( lab_{i,k,\tilde{s}_t, k} = SKE.Dec(sk_{i,k,\tilde{s}_t, k}, ct_{i,k,\tilde{s}_t, k}) \).
  6. Evaluate garbled circuit:
     a. \( (y_i, (\tilde{s}_{t+1}, \{sk_i^{i+1, k, \tilde{s}_{t+1} + 1}\}_{k \in \ell, b \in \{0, 1\}})) = GC.Eval(\tilde{U}_i, \{lab_{i,j,f[j]}\}_{j \in \ell, b \in \{0, 1\}, \{lab_{i,k,\tilde{s}_t, k}\}_{k \in \ell}) \).
  7. Dec.st\(_t+1\) = \((\tilde{s}_{t+1}, \{sk_i^{i+1, k, \tilde{s}_{t+1} + 1}\}_{k \in \ell, b \in \{0, 1\}} \).
  8. Output \((y_i, Dec.st\(_t+1\)) \).
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 One-sSE scheme on security parameter $1^\lambda$, function size $\ell_F$, state size $\ell_S$, input size $\ell_X$, and output size $\ell_Y$ are $\text{poly}(\lambda, \ell_F, \ell_S, \ell_X, \ell_Y)$.

**Correctness Intuition:** The $i$\textsuperscript{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 decryption state (or the function key if $i = 1$), we can recover the input labels corresponding to an encryption $\tilde{st}_i$ of $st_i$. Then, we can use the garbled circuit to evaluate $U_i(f, \tilde{st}_i)$ which gives us $y_i$, an encryption $\tilde{st}_{i+1}$ of $st_{i+1}$ where $f(x_i, st_i) = (y_i, st_{i+1})$, and keys for later recovering the input labels corresponding to $st_{i+1}$. This gives us the desired output.

**Correctness:** More formally, let $p$ be any polynomial and consider any $\lambda$ and any $\ell_F, \ell_S, \ell_X, \ell_Y \leq p(\lambda)$. Let $SK_f$ be a function key for function $f \in F[\ell_F, \ell_S, \ell_X, \ell_Y]$, and let $CT = \{CT_i\}_{i \in [n]}$ be a ciphertext for stream $x$ where $x = x_1 \ldots x_n$ for some $n \in [2^{\lambda}]$ and where each $x_i \in \{0,1\}^{\ell_F}$.

The function key $SK_f$ consists of
- The function $f$
- PRF\textsubscript{2} keys $\{K_{j,f[j]}\}_{j \in [\ell_F]}$ for computing SKE keys $\{sk_{i,j,f[j]}\}_{j \in [\ell_F]}$ corresponding to $f$.
- A (one-time-pad) encryption $\tilde{st}_1$ of state $st_1$
- SKE keys $\{sk_{i,k,\tilde{st}_1[k]}\}_{k \in [\ell_S]}$ corresponding to $\tilde{st}_1$.

The $i$\textsuperscript{th} ciphertext $CT_i$ consists of
- A garbled circuit $\tilde{U}_i$ for the circuit $U[x_i, p_i, p_{i+1}, \{sk_{i+1,k,b}\}_{k \in [\ell_S], b \in \{0,1\}}]$ depicted in Figure 8.
- Ciphertexts $ct_{i,j,b}$ encrypting labels $lab_{i,j,b}$ for each $j \in [\ell_F], b \in \{0,1\}$.
- Ciphertexts $ct'_{i,k,b}$ encrypting labels $lab'_{i,k,b}$ for each $k \in [\ell_S], b \in \{0,1\}$.

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

$$\text{One-sSE.Dec}(SK_f, \text{Dec.st}_i, i, CT_i)$$

$$= \text{GC.Eval}(\tilde{U}_i, \{\text{SKE.Dec}(sk_{i,j,f[j]}, ct_{i,j,f[j]})\}_{j \in [\ell_F]}, \{\text{SKE.Dec}(sk_{i,k,\tilde{st}_i[k]}, lab'_{i,k,\tilde{st}_i[k]})\}_{k \in [\ell_S]})$$

$$= \text{GC.Eval}(\tilde{U}_i, \{lab_{i,j,f[j]}\}_{j \in [\ell_F]}, \{lab'_{i,k,\tilde{st}_i[k]}\}_{k \in [\ell_S]})$$

$$= U[x_i, p_i, p_{i+1}, \{sk_{i+1,k,b}\}_{k \in [\ell_S], b \in \{0,1\}}](f, \tilde{st}_i)$$

$$= (y_i, \text{Dec.st}_i = (\tilde{st}_{i+1}, \{sk'_{i+1,k,\tilde{st}_{i+1}[k]}\}_{k \in [\ell_S]}))$$

where $(y_i, st_{i+1}) = f(x_i, st_i)$, $\tilde{st}_{i+1}$ is a (one-time-pad) encryption of $st_{i+1}$, and $\{sk_{i+1,k,\tilde{st}_{i+1}[k]}\}_{k \in [\ell_S]}$ are SKE keys corresponding to $\tilde{st}_{i+1}$. Here, the first equality follows from the definition of One-sSE.Dec and our use of PRF\textsubscript{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 One-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.

- **Hybrid**$_{0}$(λ): This is the real world experiment where we use the algorithms defined in our construction.

- **Hybrid**$_{1}$(λ): 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.

- **Hybrid**$_{2}$(λ): The outputs generated by the PRF keys $K$, $K'$ and $K_p$ are replaced with truly random strings. Indistinguishability follows from the security of PRF$_1$, PRF$_3$, and PRF$_p$.

- **Hybrid**$_{3}$(λ): For each index $i$, we change how $st_i$ and $p_i$ are generated. In the previous hybrids, we sampled a random $p_i$ and set $st_i = st_i \oplus p_i$. Now, we sample a random $st_i$ and set $p_i = st_i \oplus st_j$. Since we have just swapped the roles of two random variables in an XOR equation, the two hybrids are identically distributed.

- **Hybrid**$_{4}$(λ): 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 $\{K_{j,f}[j]\}_{j \in [\ell_F]}$ and does not sample keys $\{K_{j,1-f[j]}\}_{j \in [\ell_F]}$. When answering the challenge message queries, the challenger samples SKE keys sk$_{i,j,1-f[j]}$, for $j \in [\ell_F]$, by using true randomness instead of using $K_{j,1-f[j]}$ as was done previously. Indistinguishability follows by the security of PRF$_2$.

- **Hybrid**$_{5}$(λ): We replace the ciphertexts ct$_{i,j,1-f[j]}$ with encryptions of ⊥. 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 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 $\alpha \in [\text{Bound}_A]$ where Bound$_A$ is a bound on the runtime of $A$, and thus an implicit bound on the number of challenge message queries made by the adversary. On iteration $\alpha$, the goal is to switch to a hybrid where we simulate the $\alpha^{th}$ garbled circuit.

  - **Hybrid**$_{6,\alpha,0}$(λ): For $i < \alpha$, we generate the garbled circuit $\tilde{U}_i$ and the input labels corresponding to $(f, \tilde{st}_i)$ 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 sk$_{\alpha,k,1-\tilde{st}[\alpha]}$ into $U_{\alpha-1}$. Thus, we can replace the ciphertexts ct'$_{\alpha,k,1-\tilde{st}[\alpha]}$ with encryptions of ⊥. This removes the input labels corresponding to the negation of the bit-string for $\tilde{st}_\alpha$ from the adversary’s view. For $\alpha = 1$, this hybrid is indistinguishable from **Hybrid**$_{5}$(λ) by the security of SKE.
- **Hybrid}_{6,\alpha,1}(\lambda)**: We generate the garbled circuit $\tilde{U}_\alpha$ and the input labels corresponding to $(f, \tilde{s}_\alpha)$ using the simulator for the garbling scheme. This is feasible because we have already removed the $\alpha^{th}$ input labels for both the negation of the bit-string for $f$ and the negation of the bit-string for $\tilde{s}_i$ from the adversary’s view. Indistinguishability follows from the security of the garbling scheme. Additionally, the indistinguishability of $\text{Hybrid}_{6,\alpha,1}^A$ and $\text{Hybrid}_{6,\alpha+1,0}^A$ follows from the security of SKE.

- **Hybrid}_{7}(\lambda)**: This is the ideal world experiment written using an explicit simulator $\text{Sim}$. This hybrid is identically distributed to $\text{Hybrid}_{6,\text{Bound},1}^A$ since $\text{Hybrid}_{6,\text{Bound},1}^A$ 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.

Hybrid$_0^A(1^\lambda)$: This is the real world experiment.

1. **Parameters**: The adversary $A$ receives security parameter $1^\lambda$, and outputs a function size $1^{\ell_x}$, a state size $1^{\ell_s}$, an input size $1^{\ell_x}$, and an output size $1^{\ell_y}$.

2. **Setup**: $K \leftarrow \text{PRF}_1.\text{Setup}(1^\lambda)$, $K' \leftarrow \text{PRF}_3.\text{Setup}(1^\lambda)$, $K_p \leftarrow \text{PRF}_p.\text{Setup}(1^\lambda)$.

3. **Function Query**:
   a. $A$ outputs a streaming function query $f \in \mathcal{F}[\ell_x, \ell_s, \ell_x', \ell_y]$.
   b. **Compute PRF keys for generating SKE keys for $f$**:
      i. Compute $\{K_{j,f}[\ell]\}_{j \in \ell_x}$ from $K$.
   c. **Compute SKE keys for $\tilde{st}_1$**:
      i. Compute $p_1$ from $K_p$.
      ii. $\tilde{st}_1 = p_1$.
      iii. Compute $\{sk'_{1,k,\tilde{st}_1[k]}\}_{k \in \ell_s}$ from $K'$.
   d. Send $SK_f = (f, \{K_{j,f}[\ell]\}_{j \in \ell_x}, (\tilde{st}_1, \{sk'_{1,k,\tilde{st}_1[k]}\}_{k \in \ell_s}))$ to $A$.

4. **Challenge Message Queries**: For $i = 1, 2, 3, \ldots$
   a. $A$ outputs a challenge message $x_i \in \{0,1\}^{\ell_x}$.
   b. **Garble circuit**:
      i. Compute $p_i, p_{i+1}, \{sk'_{i+1,k,b}\}_{k \in \ell_s, b \in \{0,1\}}$ from $K', K_p$.
      ii. Let $U_i = U[x_i, p_i, p_{i+1}, \{sk'_{i+1,k,b}\}_{k \in \ell_s, b \in \{0,1\}}]$ as defined in Figure 8.
      iii. $(\tilde{U}_i, \{\text{lab}_{i,j,b}\}_{j \in \ell_x, b \in \{0,1\}}, \{\text{lab}'_{i,k,b}\}_{k \in \ell_s, b \in \{0,1\}}) \leftarrow \text{GC.Garble}(1^\lambda, U_i)$.
   c. **Encrypt function labels**: For $j \in \ell_x$, $b \in \{0,1\}$,
      i. Compute $sk_{i,j,b}$ from $K$.
      ii. $ct_{i,j,b} \leftarrow \text{SKE.Enc}(sk_{i,j,b}, \text{lab}_{i,j,b})$.
   d. **Encrypt state labels**: For $k \in \ell_s$, $b \in \{0,1\}$,
      i. Compute $sk'_{i,k,b}$ from $K'$.
      ii. $ct'_{i,k,b} \leftarrow \text{SKE.Enc}(sk'_{i,k,b}, \text{lab}'_{i,k,b})$.
   e. Send $CT_i = (\tilde{U}_i, \{ct_{i,j,b}\}_{j \in \ell_x, b \in \{0,1\}}, \{ct'_{i,k,b}\}_{k \in \ell_s, b \in \{0,1\}})$ to $A$.

5. **Experiment Outcome**: $A$ outputs a bit $b'$ which is the output of the experiment.
Hybrid$^4_{\lambda}$: We reorder several steps of the hybrid.

1. **Parameters**: The adversary $A$ receives security parameter $1^\lambda$, and outputs a function size $1^{\ell_F}$, a state size $1^{\ell_S}$, an input size $1^{\ell_X}$, and an output size $1^{\ell_Y}$.

2. **Setup**: $K \leftarrow \text{PRF}_1.\text{Setup}(1^\lambda)$, $K' \leftarrow \text{PRF}_3.\text{Setup}(1^\lambda)$, $K_p \leftarrow \text{PRF}_p.\text{Setup}(1^\lambda)$.

3. **Function Query**: 
   (a) $A$ outputs a streaming function query $f \in F[\ell_F, \ell_S, \ell_X, \ell_Y]$.

   (b) **Compute keys and pads**:
      i. $s_1 = p_1 = \text{PRF}_p.\text{Eval}(K_p, 1)$.
      ii. For $j \in [\ell_F]$, $b \in \{0,1\}$, $K_{j,b} = \text{PRF}_2.\text{Setup}(1^\lambda; \text{PRF}_1.\text{Eval}(K, (j, b)))$.
      iii. For $k \in [\ell_S]$, $b \in \{0,1\}$, $sk_{1,k,b} = \text{SKE}.\text{Setup}(1^\lambda; \text{PRF}_3.\text{Eval}(K', (1, k, b)))$.

   (c) Send $SK_f = (f, \{K_{j,j[b]}\}_{j \in [\ell_F]}, (s_1, \{sk_{1,k,b}\}_{k \in [\ell_S]})$) to $A$.

4. **Challenge Message Queries**: For $i = 1, 2, 3, \ldots$
   (a) $A$ outputs a challenge message $x_i \in \{0,1\}^{\ell_X}$.

   (b) **Compute keys and pads**:
      i. $p_{i+1} = \text{PRF}_p.\text{Eval}(K_p, i + 1)$.
      ii. For $j \in [\ell_F]$, $b \in \{0,1\}$, $sk_{i,j,b} = \text{SKE}.\text{Setup}(1^\lambda; \text{PRF}_2.\text{Eval}(K_{j,b}, i))$.
      iii. For $k \in [\ell_S]$, $b \in \{0,1\}$, $sk'_{i+1,k,b} = \text{SKE}.\text{Setup}(1^\lambda; \text{PRF}_3.\text{Eval}(K', (i + 1, k, b)))$.

   (c) $s_t = 0^{\ell_S}$.

   (d) **Compute** $(y_i, s_{i+1})$:
      i. $(y_i, s_{i+1}) = f(x_i, s_i)$.
      ii. $s_{i+1} = s_{i+1} \oplus p_{i+1}$.

   (e) **Garble circuit**:
      i. Compute $p_t, p_{t+1}, \{sk'_{i+1,k,b}\}_{k \in [\ell_S], b \in \{0,1\}}$ from $K'_{t}, K_p$.
      ii. Let $U_t = U[x_i, p_t, p_{t+1}, \{sk'_{i+1,k,b}\}_{k \in [\ell_S], b \in \{0,1\}}]$ as defined in Figure 8.
      iii. $(\bar{U}_t, \{\text{lab}_{i,j,b}\}_{j \in [\ell_F], b \in \{0,1\}}, \{\text{lab}'_{i,k,b}\}_{k \in [\ell_S], b \in \{0,1\}}) \leftarrow \text{GC}.\text{Garble}(1^\lambda, U_t)$.

   (f) **Encrypt function labels**: For $j \in [\ell_F], b \in \{0,1\}$,
      i. Compute $sk_{i,j,b}$ from $K$.
      ii. $ct_{i,j,b} \leftarrow \text{SKE}.\text{Enc}(sk_{i,j,b}, \text{lab}_{i,j,b})$.

   (g) **Encrypt state labels**: For $k \in [\ell_S], b \in \{0,1\}$,
      i. Compute $sk'_{i,k,b}$ from $K'_{t}$.
      ii. $ct'_{i,k,b} \leftarrow \text{SKE}.\text{Enc}(sk'_{i,k,b}, \text{lab}'_{i,k,b})$.

   (h) Send $CT_t = (\bar{U}_t, \{ct_{i,j,b}\}_{j \in [\ell_F], b \in \{0,1\}}, \{ct'_{i,k,b}\}_{k \in [\ell_S], b \in \{0,1\}})$ to $A$.

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

**Lemma 4.3.** For all adversaries $A$,

$$\left| \Pr[\text{Hybrid}_{\lambda}^0(1^\lambda)] - \Pr[\text{Hybrid}_{\lambda}^4(1^\lambda)] \right| = 0.$$

**Proof.** The hybrids are identical. □
**Hybrid**\(^2\)(\(1^\lambda\)): We exchange the randomness generated by \(K, K', K_p\) with true randomness.

1. **Parameters:** The adversary \(A\) receives security parameter \(1^\lambda\), and outputs a function size \(1^{\ell_F}\), a state size \(1^{\ell_S}\), an input size \(1^{\ell_X}\), and an output size \(1^{\ell_Y}\).

2. **Setup:** \(K \leftarrow \text{PRF}_1.\text{Setup}(1^\lambda), K' \leftarrow \text{PRF}_3.\text{Setup}(1^\lambda), K_p \leftarrow \text{PRF}_p.\text{Setup}(1^\lambda)\)

3. **Function Query:**
   
   (a) \(A\) outputs a streaming function query \(f \in \mathcal{F}[\ell_F, \ell_S, \ell_X, \ell_Y]\).

   (b) Compute keys and pads:
   
   i. \(\tilde{st}_1 = p_1 \leftarrow \{0, 1\}^{\ell_S}\).
   
   ii. For \(j \in [\ell_F], b \in \{0, 1\}, K_{j,b} \leftarrow \text{PRF}_2.\text{Setup}(1^\lambda)\).
   
   iii. For \(k \in [\ell_S], b \in \{0, 1\}, \sk'_1, k, b \leftarrow \text{SKE.Setup}(1^\lambda)\).

   (c) Send \(\sk_f = (f, \{K_{j,f[j]}\}_{j \in [\ell_F]}, (\tilde{st}_1, \{\sk'_1, k, \tilde{st}_1[k]\}_{k \in [\ell_S]})\) to \(A\).

4. **Challenge Message Queries:** For \(i = 1, 2, 3, \ldots\)

   (a) \(A\) outputs a challenge message \(x_i \in \{0, 1\}^{\ell_X}\).

   (b) Compute keys and pads:
   
   i. \(p_{i+1} \leftarrow \{0, 1\}^{\ell_S}\).
   
   ii. For \(j \in [\ell_F], b \in \{0, 1\}, \sk_{i,j,b} = \text{SKE.Setup}(1^\lambda; \text{PRF}_2.\text{Eval}(K_{j,b}, i))\).
   
   iii. For \(k \in [\ell_S], b \in \{0, 1\}, \sk'_{i+1, k, b} \leftarrow \text{SKE.Setup}(1^\lambda)\).

   (c) \(st_1 = 0^{\ell_S}\).

   (d) Compute \((y_i, \tilde{st}_{i+1})\):
   
   i. \((y_i, \tilde{st}_{i+1}) = f(x_i, st_i)\).
   
   ii. \(\tilde{st}_{i+1} = st_{i+1} \oplus p_{i+1}\).

   (e) Garble circuit:
   
   i. Let \(U_i = U[x_i, p_i, p_{i+1}, \{\sk'_{i+1, k, b}\}_{k \in [\ell_S], b \in \{0, 1\}}]\) as defined in Figure 8.
   
   ii. \((\tilde{U}_i, \{\lab_{i,j,b}\}_{j \in [\ell_F], b \in \{0, 1\}}, \{\lab'_{i, k, b}\}_{k \in [\ell_S], b \in \{0, 1\}}) \leftarrow \text{GC.Garble}(1^\lambda, U_i)\).

   (f) Encrypt function labels: For \(j \in [\ell_F], b \in \{0, 1\}\),

   i. \(ct_{i,j,b} \leftarrow \text{SKE.Enc}((\sk_{i,j,b}, \lab_{i,j,b})\).

   (g) Encrypt state labels: For \(k \in [\ell_S], b \in \{0, 1\}\),

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

   (h) Send \(CT_i = (\tilde{U}_i, \{ct_{i,j,b}\}_{j \in [\ell_F], b \in \{0, 1\}}, \{ct'_{i,k,b}\}_{k \in [\ell_S], b \in \{0, 1\}})\) to \(A\).

5. **Experiment Outcome:** \(A\) outputs a bit \(b'\) which is the output of the experiment.

**Lemma 4.4.** If \(\text{PRF}_1, \text{PRF}_3\) and \(\text{PRF}_p\) are secure PRFs, then for all PPT adversaries \(A\),

\[
\left| \Pr[\text{Hybrid}^4_1(1^\lambda)] - \Pr[\text{Hybrid}^4_2(1^\lambda)] \right| \leq \text{negl}(\lambda).
\]
Proof. We will first show indistinguishability between $\text{Hybrid}_1^A$ and an intermediate hybrid $\text{Hybrid}_{1,1}^A(1^\lambda)$ which is the same as $\text{Hybrid}_1^A$ 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 $A$ such that

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

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

Observe that if $B$’s PRF$_1$ oracle was a uniform random function $R$, then $B$ exactly emulates $\text{Hybrid}_{1,1}^A$, and if $B$’s PRF$_1$ oracle was PRF$_1.\text{Eval}(K, \cdot)$ for some PRF$_1$ key $K$, then $B$ emulates $\text{Hybrid}_1^A$. Moreover, $B$ 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 $B$ breaks the security of PRF$_1$ since $B$ 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 $\text{Hybrid}_2^A$. \qed
**Lemma 4.5.** For each $i$, we now determine $p_i$ by XOR-ing the true state $st_i$ with a random value $\widehat{st}_i$.

1. **Parameters:** The adversary $\mathcal{A}$ receives security parameter $1^\lambda$, and outputs a function size $1^{\ell x}$, a state size $1^{\ell s}$, an input size $1^{\ell x}$, and an output size $1^{\ell y}$.

2. **Function Query:**
   
   (a) $\mathcal{A}$ outputs a streaming function query $f \in \mathcal{F}[\ell_x, \ell_s, \ell_x, \ell_y]$.
   
   (b) **Compute keys and pads:**
   
   i. $p_1 = \widehat{st}_1 \leftarrow \{0, 1\}^{\ell s}$.
   
   ii. For $j \in [\ell_x], b \in \{0, 1\}$, $K_{j,b} \leftarrow \text{PRF}_2.\text{Setup}(1^\lambda)$.
   
   iii. For $k \in [\ell_s], b \in \{0, 1\}$, $\text{sk}_{1,k,b} \leftarrow \text{SKE}\text{.Setup}(1^\lambda)$.
   
   (c) Send $SK_f = (f, \{K_{j,f[j]}\}_{j \in [\ell_x]}, (\widehat{st}_1, \{\text{sk}_{1,k,\widehat{st}_1[k]}\}_{k \in [\ell_s]})$ to $\mathcal{A}$.

3. **Challenge Message Queries:** For $i = 1, 2, 3, \ldots$
   
   (a) $\mathcal{A}$ outputs a challenge message $x_i \in \{0, 1\}^{\ell x}$.
   
   (b) **Compute keys and pads:**
   
   i. $\widehat{st}_{i+1} \leftarrow \{0, 1\}^{\ell s}$.
   
   ii. For $j \in [\ell_x], b \in \{0, 1\}$, $\text{sk}_{i,j,b} = \text{SKE}\text{.Setup}(1^\lambda; \text{PRF}_2.\text{Eval}(K_{j,b}, i))$.
   
   iii. For $k \in [\ell_s], b \in \{0, 1\}$, $\text{sk}_{i+1,k,b} \leftarrow \text{SKE}\text{.Setup}(1^\lambda)$.
   
   (c) $st_1 = 0^{\ell s}$.
   
   (d) **Compute** $(y_i, p_{i+1})$:
   
   i. $(y_i, st_{i+1}) = f(x_i, st_i)$.
   
   ii. $p_{i+1} = st_{i+1} \oplus \widehat{st}_{i+1}$.

4. **Garble circuit:**
   
   i. Let $U_i = U[x_i, p_i, p_{i+1}, \{\text{sk}_{i+1,k,b}\}_{k \in [\ell_s], b \in \{0, 1\}}]$ as defined in Figure 8.
   
   ii. $(\widehat{U}_i, \{\text{lab}_{i,j,b}\}_{j \in [\ell_x], b \in \{0, 1\}}, \{\text{lab}_{i,k,b}\}_{k \in [\ell_s], b \in \{0, 1\}}) \leftarrow \text{GC}\text{.Garble}(1^\lambda, U_i)$.

5. **Encrypt function labels:** For $j \in [\ell_x], b \in \{0, 1\}$,
   
   i. $ct_{i,j,b} \leftarrow \text{SKE}\text{.Enc}(\text{sk}_{i,j,b}, \text{lab}_{i,j,b})$.

6. **Encrypt state labels:** For $k \in [\ell_s], b \in \{0, 1\}$,
   
   i. $ct_{i,k,b} \leftarrow \text{SKE}\text{.Enc}(\text{sk}_{i,k,b}, \text{lab}_{i,k,b})$.

(h) Send $CT_i = (\widehat{U}_i, \{ct_{i,j,b}\}_{j \in [\ell_x], b \in \{0, 1\}}, \{ct_{i,k,b}\}_{k \in [\ell_s], b \in \{0, 1\}})$ 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 $\mathcal{A}$,

$$\left| \Pr[\text{Hybrid}_3^4(1^\lambda)] - \Pr[\text{Hybrid}_2^4(1^\lambda)] \right| = 0.$$

**Proof.** The hybrids are identically distributed since we have just switched the roles of variables $p_i$ and $\widehat{st}_i$ which were uniformly distributed random variables conditioned on $st_i = p_i \oplus \widehat{st}_i$. \qed
Lemma 4.6. If PRF<sub>2</sub> is a secure PRF, then for all PPT adversaries A,

\[ \left| \Pr[\text{Hybrid}_4^A(1^\lambda)] - \Pr[\text{Hybrid}_3^A(1^\lambda)] \right| \leq \text{negl}(\lambda). \]
Proof. For $J \in [\ell_F]$, we define sub-hybrid $\text{Hybrid}_{A, J}^3(1^\lambda)$ to be the same as $\text{Hybrid}_3$ except that for $j \leq J$, we do not compute $K_{j,1-f[j]}$ during the hybrid and thus sample the corresponding keys $sk_{i,j,1-f[j]} \leftarrow \text{SKE.Setup}(1^\lambda)$ for each $i$ using uniform randomness. In other words, with regards to PRF$_2$, sub-hybrid $\text{Hybrid}_{A, J}^3(1^\lambda)$ behaves identically to $\text{Hybrid}_3^A(1^\lambda)$ for $j \leq J$, and $\text{Hybrid}_{3,J}^A(1^\lambda)$ behaves identically to $\text{Hybrid}_3^A(1^\lambda)$ for $j > J$. Observe that $\text{Hybrid}_{A, 0}^3 = \text{Hybrid}_3^A$ and $\text{Hybrid}_{A, \ell_F}^3 = \text{Hybrid}_4^A$.

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

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

We build a PPT adversary $B$ that breaks the security of PRF$_2$. $B$ follows the steps of $\text{Hybrid}_{A, J}^3$ by interacting with $A$ except that on each challenge message query $x_i$, $B$ computes $sk_{i,J,1-f[J]} \leftarrow \text{SKE.Setup}(1^\lambda, r_{i,J})$ where $r_{i,J}$ is the output of $B$’s PRF$_2$ oracle on input $i$. $B$ outputs whatever $A$ outputs.

Observe that if $B$’s PRF$_2$ oracle was a uniform random function $R$, then $B$ exactly emulates $\text{Hybrid}_{A, J-1}^3$, and if $B$’s PRF$_2$ oracle was PRF$_2.\text{Eval}(K_{j,1-f[j]}, \cdot)$ for some PRF$_2$ key $K_{j,1-f[j]}$, then $B$ emulates $\text{Hybrid}_{A, J}^3$. Moreover, $B$ 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 $B$ breaks the security of PRF$_2$ since $B$ has a non-negligible advantage in distinguishing between the two challenge oracles in the PRF$_2$ experiment.

$\square$
**Hybrid**$^4_5(\lambda)$: We replace the ciphertexts $ct_{i,j,1−f\{j\}}$ with encryptions of $\bot$. 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 $1^\lambda$, and outputs a function size $1^{\ell_F}$, a state size $1^{\ell_S}$, an input size $1^{\ell_X}$, and an output size $1^{\ell_Y}$.

2. **Function Query:**
   (a) $\mathcal{A}$ outputs a streaming function query $f \in \mathcal{F}[\ell_F, \ell_S, \ell_X, \ell_Y]$.
   (b) **Compute keys and pads:**
      i. $p_1 = \tilde{st}_1 \leftarrow \{0,1\}^{\ell_S}$.
      ii. For $j \in [\ell_F]$, $K_{j,f\{j\}} \leftarrow \text{PRF}_2.\text{Setup}(1^\lambda)$.
      iii. For $k \in [\ell_S]$, $b \in \{0,1\}$, $sk'_{i,k,b} \leftarrow \text{SKE.\text{Setup}(1^\lambda)}$.
   (c) Send $SK_f = (f, \{K_{j,f\{j\}}\}_{j \in [\ell_F]}, (\tilde{st}_1, \{sk'_{i,k,\tilde{st}_1[k]}\}_{k \in [\ell_S]}))$ to $\mathcal{A}$.

3. **Challenge Message Queries:** For $i = 1, 2, 3, \ldots$
   (a) $\mathcal{A}$ outputs a challenge message $x_i \in \{0,1\}^{\ell_X}$.
   (b) **Compute keys and pads:**
      i. $\tilde{st}_{i+1} \leftarrow \{0,1\}^{\ell_S}$.
      ii. For $j \in [\ell_F]$, $b \in \{0,1\}$,
         A. If $b = f[j]$, $sk_{i,j,b} = \text{SKE.\text{Setup}(1^\lambda; \text{PRF}_2.\text{Eval}(K_{j,b}, i))}$.
         B. If $b \neq f[j]$, $sk_{i,j,b} \leftarrow \text{SKE.\text{Setup}(1^\lambda)}$.
      iii. For $k \in [\ell_S]$, $b \in \{0,1\}$, $sk'_{i+1,k,b} \leftarrow \text{SKE.\text{Setup}(1^\lambda)}$.
   (c) $st_1 = 0^{\ell_S}$.
   (d) **Compute** $(y_i, p_{i+1})$:
      i. $(y_i, st_{i+1}) = f(x_i, st_i)$.
      ii. $p_{i+1} = st_{i+1} \oplus \tilde{st}_{i+1}$.
   (e) **Garble circuit:**
      i. Let $U_i = U[x_i, p_i, p_{i+1}, \{sk'_{i+1,k,b}\}_{k \in [\ell_S], b \in \{0,1\}}]$ as defined in Figure 8.
      ii. $(\tilde{U}_i, \{\text{lab}_{i,j,b}\}_{j \in [\ell_F], b \in \{0,1\}}, \{\text{lab}'_{i,k,b}\}_{k \in [\ell_S], b \in \{0,1\}}) \leftarrow \text{GC.\text{Garble}(1^\lambda, U_i)}$.
   (f) **Encrypt function labels:** For $j \in [\ell_F]$, $b \in \{0,1\}$,
      i. If $b = f[j]$, $ct_{i,j,b} \leftarrow \text{SKE.\text{Enc}(sk_{i,j,b}, \text{lab}_{i,j,b})}$.
      ii. If $b \neq f[j]$, $ct_{i,j,b} \leftarrow \text{SKE.\text{Enc}(sk_{i,j,b}, \bot)}$.
   (g) **Encrypt state labels:** For $k \in [\ell_S]$, $b \in \{0,1\}$,
      i. $ct'_{i,k,b} \leftarrow \text{SKE.\text{Enc}(sk'_{i,k,b}, \text{lab}'_{i,k,b})}$.
   (h) Send $CT_i = (\tilde{U}_i, \{ct_{i,j,b}\}_{j \in [\ell_F], b \in \{0,1\}}, \{ct'_{i,k,b}\}_{k \in [\ell_S], b \in \{0,1\}})$ to $\mathcal{A}$.

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

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

$$\left| \text{Pr}[\text{Hybrid}^4_5(1^\lambda)] - \text{Pr}[\text{Hybrid}^4_5(1^\lambda)] \right| \leq \text{negl}(\lambda).$$
Proof. For the sake of contradiction, assume that there exists a PPT adversary \( A \) such that

\[
\text{Adv}_A(\lambda) = \left| \Pr[\text{Hybrid}_4^A(1^\lambda)] - \Pr[\text{Hybrid}_5^A(1^\lambda)] \right|
\]

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

For each \( I \in [n], J \in [\ell_F] \), let

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

Then, by Equation 3, since

- \( \text{Hybrid}_{4, 1, 0}^A(1^\lambda) = \text{Hybrid}_4^A(1^\lambda) \),
- \( \text{Hybrid}_{5, 1, 0}^A(1^\lambda) = \text{Hybrid}_4^A(n, \ell_F, 1^\lambda) \),
- for all \( i \in [n] \), \( \text{Hybrid}_{4, I-1, \ell_F}^A = \text{Hybrid}_{4, I, 0}^A \),

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

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

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

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

**Hybrid$_{\alpha,0}(\lambda)$**: We replace the ciphertexts $ct'_{\alpha,k,1-\tilde{s}[\alpha]}$ with encryptions of $\bot$. This removes the input labels which don’t correspond to $\tilde{s}[\alpha]$ from the adversary’s view.

1. **Parameters**: The adversary $A$ receives security parameter $1^\lambda$, and outputs a function size $1^{\ell_F}$, a state size $1^{\ell_S}$, an input size $1^{\ell_X}$, and an output size $1^{\ell_Y}$.

2. **Function Query**:
   
   (a) $A$ outputs a streaming function query $f \in \mathcal{F}[\ell_F, \ell_S, \ell_X, \ell_Y]$.
   
   (b) **Compute keys and pads**:
      
      i. $p_1 = \tilde{s}_1 \leftarrow \{0,1\}^{\ell_S}$.
      
      ii. For $j \in [\ell_F]$, $K_j \leftarrow \text{PRF}_2.\text{Setup}(1^\lambda)$.
      
      iii. For $k \in [\ell_S]$, $b \in \{0,1\}$, $sk'_{1,k,b} \leftarrow \text{SKE.\text{Setup}}(1^\lambda)$.
   
   (c) Send $SK_f = (f, \{K_j\}_{j \in [\ell_F]}, (\tilde{s}_1, \{sk'_{1,k,\tilde{s}_1[k]}\}_{k \in [\ell_S]}))$ to $A$.

3. **Challenge Message Queries**: For $i = 1, 2, 3, \ldots$
   
   (a) $A$ outputs a challenge message $x_i \in \{0,1\}^{\ell_X}$.
   
   (b) **Compute keys and pads**:
      
      i. $\tilde{s}_{i+1} \leftarrow \{0,1\}^{\ell_S}$.
      
      ii. For $j \in [\ell_F]$, $b \in \{0,1\}$,
         
         A. If $b = f[j]$, $sk_{i,j,b} = \text{SKE.\text{Setup}}(1^\lambda; \text{PRF}_2.\text{Eval}(K_{j,b}, i))$.
         
         B. If $b \neq f[j]$, $sk_{i,j,b} \leftarrow \text{SKE.\text{Setup}}(1^\lambda)$.
      
      iii. For $k \in [\ell_S]$, $b \in \{0,1\}$, $sk'_{i+1,k,b} \leftarrow \text{SKE.\text{Setup}}(1^\lambda)$.
   
   (c) $s_{t_1} = 0^{\ell_S}$.
   
   (d) **Compute** $(y_i, p_{i+1})$:
      
      i. $(y_i, s_{t_{i+1}}) = f(x_i, s_{t_i})$.
      
      ii. $p_{i+1} = s_{t_{i+1}} \oplus \tilde{s}_{i+1}$.
   
   (e) **Garble circuit**:
      
      i. If $i \geq \alpha$,
         
         A. Let $U_i = U[x_i, p_i, p_{i+1}, \{sk'_{i+1,k,b}\}_{k \in [\ell_S], b \in \{0,1\}}]$ as defined in Figure 8.
         
         B. $(\tilde{U}_i, \{\text{lab}_{i,j,b}\}_{j \in [\ell_F], b \in \{0,1\}}, \{\text{lab}'_{i,k,\tilde{s}[i+1][k]}\}_{k \in [\ell_S]}) \leftarrow \text{GC.\text{Garble}}(1^\lambda, U_i)$.
      
      ii. If $i < \alpha$,
         
         A. $(\tilde{U}_i, \{\text{lab}_{i,j,f[j]}\}_{j \in [\ell_F]}, \{\text{lab}'_{i,k,\tilde{s}[i][k]}\}_{k \in [\ell_S]})$
         
         $\leftarrow \text{GC.\text{Sim}}(1^\lambda, 1^{\ell_U}, (f, \tilde{s}_i), (y_i, \{sk'_{i+1,k,\tilde{s}[i+1][k]}\}_{k \in [\ell_S]}))$
   
   (f) **Encrypt function labels**: For $j \in [\ell_F]$, $b \in \{0,1\}$,
      
      i. If $b = f[j]$, $ct_{i,j,b} \leftarrow \text{SKE.\text{Enc}}(sk_{i,j,b}, \text{lab}_{i,j,b})$.
      
      ii. If $b \neq f[j]$, $ct_{i,j,b} \leftarrow \text{SKE.\text{Enc}}(sk_{i,j,b}, \bot)$.
   
   (g) **Encrypt state labels**: For $k \in [\ell_S]$, $b \in \{0,1\}$,
i. If $i > \alpha$ or $b = \text{est}_i[k]$, $\text{ct}'_{i,k,b} \leftarrow \text{SKE.Enc}(\text{sk}'_{i,k,b}, \text{lab}'_{i,k,b})$.

ii. If $i \leq \alpha$ and $b \neq \text{est}_i[k]$, $\text{ct}'_{i,k,b} \leftarrow \text{SKE.Enc}(\text{sk}'_{i,k,b}, \bot)$.

(h) Send $CT_i = (\tilde{U}_i, \{\text{ct}_{i,j,b}\}_{j \in [\ell_F], b \in \{0,1\}}, \{\text{ct}'_{i,k,b}\}_{k \in [\ell_S], b \in \{0,1\}})$ to $A$.

4. **Experiment Outcome:** $A$ 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 $A$,

$$\left| \Pr[H_{Hybrid_5}^A(1^\lambda)] - \Pr[H_{Hybrid_6}^{A,1,0}(1^\lambda)] \right| \leq \text{negl}(\lambda).$$

**Proof.** Suppose for sake of contradiction, that there exists a PPT adversary $A$ such that

$$\left| \Pr[H_{Hybrid_5}^A(1^\lambda)] - \Pr[H_{Hybrid_6}^{A,1,0}(1^\lambda)] \right| > \text{negl}(\lambda) \quad (4)$$

We build a PPT adversary $B$ that breaks the security of $SKE$. $B$ receives the security parameter from its $SKE$ challenger and provides it to $A$ who outputs parameters $(1^{\ell_F}, 1^{\ell_S}, 1^{\ell_X}, 1^{\ell_Y})$ to $B$. When $A$ outputs a challenge function query $f$, $B$ behaves identically to $H_{Hybrid_5}^A(1^\lambda)$ except that $B$ does not compute $\text{sk}'_{1,k,1-\tilde{w}_1[k]}$. When $A$ outputs a challenge message query $x_i$, $B$ behaves identically to $H_{Hybrid_5}^A(1^\lambda)$ except that if $i = 1$, rather than computing $\text{ct}'_{1,k,1-\tilde{w}_1[k]}$ as in $H_{Hybrid_5}^A(1^\lambda)$, $B$ sends challenge message pair $(\text{lab}_{1,k,1-\tilde{w}_1[k]}, \bot)$ to its $SKE$ challenger, receives back $\text{ct}^*$ from its $SKE$ challenger, and sets $\text{ct}_{1,k,1-\tilde{w}_1[k]} = \text{ct}^*$. At the end of the experiment, $B$ outputs whatever $A$ outputs.

Observe that if $\text{ct}^*$ was an encryption of $\text{lab}_{1,k,1-\tilde{w}_1[k]}$, then $B$ exactly emulates $H_{Hybrid_5}^A$, and if $\text{ct}^*$ was an encryption of $\bot$, then $B$ emulates $H_{Hybrid_6}^{A,1,0}$. Moreover, $B$ does not need to know the $SKE$ key $\text{sk}_{1,k,1-\tilde{w}_1[k]}$ for performing these experiments. Therefore, by Equation 4, this means that $B$ breaks the security of $SKE$ since $B$ can distinguish between the two $SKE$ ciphertexts with non-negligible probability. 

$\square$
Hybrid$^4_{\alpha,\lambda}(\lambda)$: We simulate the garbled circuit for $U_\alpha$.

1. **Parameters**: The adversary $A$ receives security parameter $1^\lambda$, and outputs a function size $1^{\ell_F}$, a state size $1^{\ell_S}$, an input size $1^{\ell_X}$, and an output size $1^{\ell_Y}$.

2. **Function Query**:
   
   (a) $A$ outputs a streaming function query $f \in \mathcal{F}[\ell_F, \ell_S, \ell_X, \ell_Y]$.
   
   (b) Compute keys and pads:
      
      i. $p_1 = \tilde{st}_1 \leftarrow \{0,1\}^{\ell_S}$.
      
      ii. For $j \in \ell_F$, $K_{j,f[j]} \leftarrow \text{PRF}_2.\text{Setup}(1^\lambda)$.
      
      iii. For $k \in \ell_S$, $b \in \{0,1\}$, $\text{sk}_{i,k,b} \leftarrow \text{SKE.\text{Setup}}(1^\lambda)$.
   
   (c) $\text{Send } SK_f = (f, \{K_{j,f[j]}\}_{j \in \ell_F}, (\tilde{st}_1, \{\text{sk}_{i,k,\tilde{st}_1[k]}\}_{k \in \ell_S}))$ to $A$.

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

   (a) $A$ outputs a challenge message $x_i \in \{0,1\}^{\ell_X}$.
   
   (b) Compute keys and pads:
       
       i. $\tilde{st}_{i+1} \leftarrow \{0,1\}^{\ell_S}$.
       
       ii. For $j \in \ell_F$, $b \in \{0,1\}$,
           
           A. If $b = f[j]$, $\text{sk}_{i,j,b} \leftarrow \text{SKE.\text{Setup}}(1^\lambda; \text{PRF}_2.\text{Eval}(K_{j,b}, i))$.
           
           B. If $b \neq f[j]$, $\text{sk}_{i,j,b} \leftarrow \text{SKE.\text{Setup}}(1^\lambda)$.
       
       iii. For $k \in \ell_S$, $b \in \{0,1\}$, $\text{sk}'_{i+1,k,b} \leftarrow \text{SKE.\text{Setup}}(1^\lambda)$.
   
   (c) $\text{st}_1 = 0^{\ell_S}$.

   (d) Compute $(y_i, p_{i+1})$:
       
       i. $(y_i, \text{st}_{i+1}) = f(x_i, \text{st}_i)$.
       
       ii. $p_{i+1} = \text{st}_{i+1} \oplus \text{st}_{i+1}$.

   (e) **Garble circuit**:
       
       i. If $i > \alpha$,
           
           A. Let $U_i = U[x_i, p_i, p_{i+1}, \{\text{sk}'_{i+1,k,b}\}_{k \in \ell_S}, b \in \{0,1\}]$ as defined in Figure 8.
           
           B. $(\tilde{U}_i, \{\text{lab}_{i,j,b}\}_{j \in \ell_F}, b \in \{0,1\}, \{\text{lab}'_{i,k,b}\}_{k \in \ell_S}, b \in \{0,1\}) \leftarrow \text{GC.\text{Garble}}(1^\lambda, U_i)$.
       
       ii. If $i \leq \alpha$,
           
           A. $(\tilde{U}_i, \{\text{lab}_{i,j,f[j]}\}_{j \in \ell_F}, \{\text{lab}'_{i,k,\tilde{st}_1[k]}\}_{k \in \ell_S})$
           
           $\leftarrow \text{GC.\text{Sim}}(1^\lambda, 1^{\ell_U}(f, \tilde{st}_i), (y_i, (\tilde{st}_{i+1}, \{\text{sk}'_{i+1,k,\tilde{st}_1[k]}\}_{k \in \ell_S})))$.
   
   (f) **Encrypt function labels**: For $j \in \ell_F$, $b \in \{0,1\}$,
       
       i. If $b = f[j]$, $\text{ct}_{i,j,b} \leftarrow \text{SKE.\text{Enc}}(\text{sk}_{i,j,b}, \text{lab}_{i,j,b})$.
       
       ii. If $b \neq f[j]$, $\text{ct}_{i,j,b} \leftarrow \text{SKE.\text{Enc}}(\text{sk}_{i,j,b}, \bot)$.

   (g) **Encrypt state labels**: For $k \in \ell_S$, $b \in \{0,1\}$,
       
       i. If $i > \alpha$ or $b = \tilde{st}_1[k]$, $\text{ct}'_{i,k,b} \leftarrow \text{SKE.\text{Enc}}(\text{sk}'_{i,k,b}, \text{lab}'_{i,k,b})$.
       
       ii. If $i \leq \alpha$ and $b \neq \tilde{st}_1[k]$, $\text{ct}'_{i,k,b} \leftarrow \text{SKE.\text{Enc}}(\text{sk}'_{i,k,b}, \bot)$.

   (h) Send $CT_i = (\tilde{U}_i, \{\text{ct}_{i,j,b}\}_{j \in \ell_F}, b \in \{0,1\}, \{\text{ct}'_{i,k,b}\}_{k \in \ell_S}, b \in \{0,1\})$ to $A$.

4. **Experiment Outcome**: $A$ 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 $A$,

$$\left| \Pr[\text{Hybrid}^A_{6,\alpha,0}(1^\lambda)] - \Pr[\text{Hybrid}^A_{6,\alpha,1}(1^\lambda)] \right| \leq \text{negl}(\lambda).$$

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

$$\left| \Pr[\text{Hybrid}^A_{6,\alpha,0}(1^\lambda)] - \Pr[\text{Hybrid}^A_{6,\alpha,1}(1^\lambda)] \right| > \text{negl}(\lambda) \quad (5)$$

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

Observe that if the GC challenger generates the garbled circuits and input labels honestly, then $B$ exactly emulates $\text{Hybrid}^A_{6,\alpha,0}$, and if the GC challenger simulates these values, then $B$ emulates $\text{Hybrid}^A_{6,\alpha,1}$. Thus, by Equation 5, $B$ breaks the security of GC since $B$ 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 $A$,

$$\left| \Pr[\text{Hybrid}^A_{6,\alpha,1}(1^\lambda)] - \Pr[\text{Hybrid}^A_{6,\alpha+1,0}(1^\lambda)] \right| \leq \text{negl}(\lambda).$$

Proof. This proof follows by a straightforward reduction to the security of SKE using a proof similar to Lemma 4.8. In particular, for $k \in [\ell_S]$, we can use the security of SKE to change ciphertexts $c_{\alpha+1,k,1-\bar{s}_a+1}[k]$ from encryptions of $\text{lab}'_{a+1,k,1-\bar{s}_a+1}[k]$ to encryptions of $\bot$. This can be argued since the secret keys $sk'_{a+1,k,1-\bar{s}_a+1}[k]$ are no longer used anywhere in these hybrids because the garbled circuit for $U_a$ is now being simulated. For brevity, we omit further details.

\[\square\]
**Proof.** The hybrids are identical.

**Lemma 4.11.** This is the ideal world experiment written using an explicit simulator Sim. This hybrid is identical to \( \text{Hybrid}^A_{\text{Bound},1} \) where \( \text{Bound}_A \) is a bound on the runtime of \( A \), and thus an implicit bound on the number of message queries \( A \) will make.

1. **Parameters:** The adversary \( A \) receives security parameter \( 1^\lambda \), and outputs a function size \( 1^{\ell_F} \), a state size \( 1^{\ell_S} \), an input size \( 1^{\ell_X} \), and an output size \( 1^{\ell_Y} \). The simulator Sim receives \( (1^\lambda, 1^{\ell_F}, 1^{\ell_S}, 1^{\ell_X}, 1^{\ell_Y}) \).

2. **Function Query:**
   
   (a) \( A \) outputs a streaming function query \( f \in \mathcal{F}[\ell_F, \ell_S, \ell_X, \ell_Y] \).

   (b) **Simulated Function Key:** Sim receives \( f \) and computes the following:

   i. **Compute keys and pads:**
      
      A. \( p_1 = \tilde{st}_1 \leftarrow \{0,1\}^{\ell_S} \).
      
      B. For \( j \in [\ell_F] \), \( K_{j,f[j]} \leftarrow \text{PRF}_2.\text{Setup}(1^\lambda) \).
      
      C. For \( k \in [\ell_S] \), \( b \in \{0,1\} \), \( sk_{i,k,b}' \leftarrow \text{SKE}.\text{Setup}(1^\lambda) \).
   
   ii. Send \( SK_j = (f, \{K_{j,f[j]}\}_{j \in [\ell_F]}, (\tilde{st}_1, \{sk_{i,k,\tilde{st}_1[k]}'\}_{k \in [\ell_S]}) \) to \( A \).

3. **Challenge Message Queries:** For \( i = 1, 2, 3, \ldots \)
   
   (a) \( A \) outputs a challenge message \( x_i \in \{0,1\}^{\ell_X} \).

   (b) **Compute Output Value:** \( (y_i, st_{i+1}) = f(x_i, st_i) \) where \( st_1 = 0^{\ell_S} \).

   (c) **Simulated Ciphertext:** Sim receives \( y_i \) and computes the following:
   
   i. \( \tilde{st}_{i+1} \leftarrow \{0,1\}^{\ell_S} \).
   
   ii. For \( j \in [\ell_F] \), \( b \in \{0,1\} \),
   
      A. If \( b = f[j] \), \( sk_{i,j,b} = \text{SKE}.\text{Setup}(1^\lambda; \text{PRF}_2.\text{Eval}(K_{j,b}, i)) \).
   
      B. If \( b \neq f[j] \), \( sk_{i,j,b} \leftarrow \text{SKE}.\text{Setup}(1^\lambda) \).
   
   iii. For \( k \in [\ell_S] \), \( b \in \{0,1\} \), \( sk_{i+1,k,b}' \leftarrow \text{SKE}.\text{Setup}(1^\lambda) \).
   
   iv. **Garble circuit:**
   
      A. \( (\tilde{U}_i, \{\text{lab}_{i,j,f[j]}\}_{j \in [\ell_F]}, \{\text{lab}_{i,k,\tilde{st}_i[k]}'\}_{k \in [\ell_S]}) \)
      
      \( \leftarrow \text{GC}.\text{Sim}(1^\lambda, 1^{\ell_U}, (f, \tilde{st}_i), (y_i, \{sk_{i+1,k,\tilde{st}_i[k]}'\}_{k \in [\ell_S]}) \).
   
   v. **Encrypt function labels:** For \( j \in [\ell_F] \), \( b \in \{0,1\} \),
   
      A. If \( b = f[j] \), \( ct_{i,j,b} \leftarrow \text{SKE}.\text{Enc}(sk_{i,j,b}, \text{lab}_{i,j,b}) \).
   
      B. If \( b \neq f[j] \), \( ct_{i,j,b} \leftarrow \text{SKE}.\text{Enc}(sk_{i,j,b}, \perp) \).
   
   vi. **Encrypt state labels:** For \( k \in [\ell_S] \), \( b \in \{0,1\} \),
   
      A. If \( b = \tilde{st}_i[k] \), \( ct_{i,k,b}' \leftarrow \text{SKE}.\text{Enc}(sk_{i,k,b}', \text{lab}_{i,k,b}') \).
   
      B. If \( b \neq \tilde{st}_i[k] \), \( ct_{i,k,b}' \leftarrow \text{SKE}.\text{Enc}(sk_{i,k,b}', \perp) \).
   
   vii. Send \( CT_i = (\tilde{U}_i, \{ct_{i,j,b}\}_{j \in [\ell_F], b \in \{0,1\}}, \{ct_{i,k,b}'\}_{k \in [\ell_S], b \in \{0,1\}}) \) to \( A \).

4. **Experiment Outcome:** \( A \) outputs a bit \( b' \) which is the output of the experiment.

**Lemma 4.11.** For all adversaries \( A \),

\[
\left| \Pr[\text{Hybrid}^A_{0,\text{Bound},1}(1^\lambda)] - \Pr[\text{Hybrid}^A_{\text{Bound},1}(1^\lambda)] \right| = 0.
\]

where \( \text{Bound}_A \) is a bound on the runtime of \( A \).

**Proof.** The hybrids are identical. \( \square \)
Thus, our lemmas give us the following corollary:

**Corollary 4.12.** If

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

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

**Proof.** The corollary immediately follows from Lemmas 4.3-4.11.

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 \(s\)FE 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 \(P/Poly\)
2. a single-key, single-ciphertext, function-selective-IND-secure, secret-key \(sFE\) scheme for \(P/Poly\)

there exists a \(Q\)-bounded, semi-adaptive-function-selective-IND-secure, public-key (resp. secret-key) \(sFE\) scheme for \(P/Poly\).

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 \(P/Poly\) for any polynomial \(Q = Q(\lambda)\) of the security parameter \(\lambda\).

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

**Tools.**

- \(\text{One-sFE} = (\text{One-sFE.Setup, One-sFE.EncSetup, One-sFE.Enc, One-sFE.KeyGen, One-sFE.Dec})\): A single-key, single-ciphertext, function-selective-IND-secure, secret-key \(sFE\) scheme for \(P/Poly\).

- The following primitives can be built from a \(Q\)-bounded, adaptive-IND-secure, public-key (resp. secret-key) \(FE\) scheme for \(P/Poly\):
  - \(\text{PRF} = (\text{PRF.Setup, PRF.Eval})\): A secure pseudorandom function family.
  - \(\text{PRF_2} = (\text{PRF_2.Setup, PRF_2.Eval})\): A secure pseudorandom function family.
  - \(\text{SKE} = (\text{SKE.Setup, SKE.Enc, SKE.Dec})\): A secure secret-key encryption scheme with pseudorandom ciphertexts.
  - \(\text{FE} = (\text{FE.Setup, FE.Enc, FE.KeyGen, FE.Dec})\): A \(Q\)-bounded, selective-IND-secure, public-key (resp. secret-key) \(FE\) scheme for \(P/Poly\).
  - \(\text{FPFE} = (\text{FPFE.Setup, FPFE.Enc, FPFE.KeyGen, FPFE.Dec})\): A \(Q\)-bounded, function-private, function-selective-IND-secure, secret-key \(FE\) scheme for \(P/Poly\).

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

- We can build \(\text{PRF, PRF_2, SKE}\) 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 \(\text{AdFE}\).

- \(\text{AdFE}\) already satisfies the security requirements needed for \(\text{FE}\).

- \(\text{AdFE}\) immediately implies a \(Q\)-bounded, function-selective-IND-secure, secret-key \(FE\) scheme \(\text{SKFE}\) for \(P/Poly\). We can then build \(\text{FPFE}\) by using the function-privacy transformation of [BS18] on \(\text{SKFE}\).

\(^{19}\)In particular, [AV19] show how to build a \(Q\)-bounded, adaptive-IND-secure, public-key (resp. secret-key) \(FE\) scheme for \(P/Poly\) 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 \(P/Poly\) from \(\text{OWFs}\). Note that \(\text{OWFs}\) 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 $\lambda$, function size $\ell_F$, state size $\ell_S$, input size $\ell_X$, and output size $\ell_Y$, we will instantiate our primitives with the following parameters:

<table>
<thead>
<tr>
<th>Function</th>
<th>Security Parameter</th>
<th>Input Size</th>
<th>Output Size</th>
<th>Function Size</th>
<th>State Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-sFE</td>
<td>$\lambda$</td>
<td>$\ell_X$</td>
<td>$\ell_Y$</td>
<td>$\ell_F$</td>
<td>$\ell_S$</td>
</tr>
<tr>
<td>PRF</td>
<td>$\lambda$</td>
<td>$\lambda$</td>
<td>$\lambda$</td>
<td>$5\lambda$</td>
<td>$\lambda$</td>
</tr>
<tr>
<td>PRF2</td>
<td>$\lambda$</td>
<td>$\lambda$</td>
<td>$\lambda$</td>
<td>$\lambda$</td>
<td>$\lambda$</td>
</tr>
<tr>
<td>SKE</td>
<td>$\lambda$</td>
<td>$\lambda$</td>
<td>$\lambda$</td>
<td>$\lambda$</td>
<td>$\lambda$</td>
</tr>
<tr>
<td>FPFE</td>
<td>$\lambda$</td>
<td>$\ell_{FPFE.m}$</td>
<td>$\ell_{FPFE.out}$</td>
<td>$\ell_H$</td>
<td>$\ell_G$</td>
</tr>
<tr>
<td>FE</td>
<td>$\lambda$</td>
<td>$\ell_{FE.m}$</td>
<td>$\ell_{FE.out}$</td>
<td>$\ell_H$</td>
<td>$\ell_G$</td>
</tr>
</tbody>
</table>

where we define

- $\ell_{FPFE.m} = 2\lambda + 2\ell_X + \ell_{One-sFE.ct}$ where $\ell_{One-sFE.ct}$ is the size of ciphertexts of One-sFE.
- $\ell_{FPFE.out} = \ell_{One-sFE.ct}$ where $\ell_{One-sFE.ct}$ is the size of ciphertexts of One-sFE.
- $\ell_H$ is the maximum of
  - the size of $H[One-sFE.msk, One-sFE.Enc.st, PRF2.K]$ defined in Figure 10
  - the size of $H'[One-sFE.msk, One-sFE.Enc.st, PRF2.K]$ 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 $PRF2.K$ of PRF2.

- $\ell_{FE.m} = \ell_{FPFE.msk} + \ell_{PRF.K} + 1 + \ell_{SKE.sk}$ where $\ell_{FPFE.msk}$ is the size of master secret keys of FPFE, $\ell_{PRF.K}$ is the size of keys of PRF, and $\ell_{SKE.sk}$ is the size of keys of SKE.
- $\ell_{FE.out} = \ell_{One-sFE.sk} + \ell_{FPFE.sk}$ where $\ell_{One-sFE.sk}$ is the size of secret keys of One-sFE and $\ell_{FPFE.sk}$ is the size of function keys of FPFE.
- $\ell_G$ is the maximum size of $G_{f,s,c}$ defined in Figure 9 for any $f \in F[\ell_F, \ell_S, \ell_X, \ell_Y], s \in \{0, 1\}^\lambda$, and $c$ of size $\ell_{SKE.ct}$ where $\ell_{SKE.ct}$ is the size of ciphertexts of SKE when encrypting plaintexts of size $\ell_{SKE.m} = \ell_{FE.out}$.

**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 $\lambda$, all algorithms only require randomness of length $\lambda$. If the original algorithm required additional randomness, we can replace it with a new algorithm that first expands the $\lambda$ 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 $\ell_F, \ell_S, \ell_X, \ell_Y$ are polynomial in $\lambda$).
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 setting, $sFE$ and $FE$ are both secret-key schemes, and in the public-key setting, $sFE$ and $FE$ are both public-key schemes. We use input $MSK/MPK$ 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.

- **sFE.Setup($1^\lambda, 1^{\ell_X}, 1^{\ell_s}, 1^{\ell_X}, 1^{\ell_Y}$):**
  1. **Secret-Key Setting:**
     (a) $FE.msk \leftarrow FE.Setup(1^\lambda)$.
     (b) Output $MSK = FE.msk$.
  2. **Public-Key Setting:**
     (a) $(FE.mpk, FE.msk) \leftarrow FE.Setup(1^\lambda)$.
     (b) Output $(MPK, MSK) = (FE.mpk, FE.msk)$.

- **sFE.EncSetup($MSK/MPK$):**
  1. **Secret-Key Setting:** Parse $MSK = FE.msk$. Set $FE.ek = FE.msk$.
  2. **Public-Key Setting:** Parse $MPK = FE.mpk$. Set $FE.ek = FE.mpk$.
  3. $PRF.K \leftarrow PRF.SetUp(1^\lambda)$.
  4. $FPFE.msk \leftarrow FPFE.SetUp(1^\lambda)$.
  5. $FE.ct \leftarrow FE.Enc(FE.ek, (FPFE.msk, PRF.K, 0, 0^{\ell_{SKE.\alpha}}))$.
  6. Output $Enc.ST = (FPFE.msk, FE.ct)$.

- **sFE.Enc($MSK/MPK$, $Enc.ST$, $i$, $x_i$):**
  1. Parse $Enc.ST = (FPFE.msk, FE.ct)$.
  2. $t_i \leftarrow \{0, 1\}^\lambda$.
  3. $FPFE.ct_i \leftarrow FPFE.Enc(FPFE.msk, (i, t_i, x_i, 0^{\ell_X}, 0^{\ell_{One-sFE.\alpha}}))$.
  4. If $i = 1$, output $CT_1 = (FE.ct, FPFE.ct_1)$.
  5. Else, output $CT_i = FPFE.ct_i$.

- **sFE.KeyGen($MSK$, $f$):**
  1. Parse $MSK = FE.msk$.
  2. $s \leftarrow \{0, 1\}^\lambda$.
  3. $c \leftarrow \{0, 1\}^{\ell_{SKE.\alpha}}$.
  4. Let $G = G[f, s, c]$ as defined in Figure 9.
  5. $FE.sk_G \leftarrow FE.KeyGen(FE.msk, G)$.
We also define the following functions which will be used in our security proof.

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

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

2. Else,
   
   (a) Output \((\text{One-sFE}.\text{sk}_f, \text{FPFE}.\text{sk}_H) \leftarrow \text{SKE}.\text{Dec}(\text{SKE}.\text{sk}, c).\)

Figure 9: Definition of \( G[f, s, c] \)

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

1. \( r_i \leftarrow \text{PRF2}.\text{Eval}(\text{PRF2}.K, t_i).\)

2. Output \((\text{One-sFE}.\text{ct}_i \leftarrow \text{One-sFE}.\text{Enc}(\text{One-sFE}.\text{msk}, \text{One-sFE}.\text{Enc.st}, i, x_i; r_i).\)

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

- \text{sFE}.\text{Dec}(\text{SK}_f, \text{Dec.ST}_i, i, CT_i):

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

We also define the following functions which will be used in our security proof.
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 $ℓ_ω$, state size $ℓ_s$, input size $ℓ_X$, and output size $ℓ_y$ are $\text{poly}(λ, ℓ_ω, ℓ_s, ℓ_X, ℓ_y)$.

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

**Correctness:** More formally, let $p$ be any polynomial and consider any $λ$ and any $ℓ_ω, ℓ_s, ℓ_X, ℓ_Y \leq p(λ)$. Let $\text{SK}_f$ be a function key for function $f \in \mathcal{F}[ℓ_ω, ℓ_s, ℓ_X, ℓ_Y]$, and let $\text{CT} = \{\text{CT}_i\}_{i \in [n]}$ be a ciphertext for $x$ where $x = x_1 \ldots x_n$ for some $n \in [2^λ]$ and where each $x_i \in \{0, 1\}^{ℓ_X}$.

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

$$\text{FE.Dec(\text{FE.sk}_G, \text{FE.ct})} = G[f, s, c](\text{FPFE.mpk}, \text{PRF.\text{K}}, 0, 0^{\text{SK}_f}) = (\text{One-sFE.sk}_f, \text{FPFE.sk}_H)$$

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

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

$\text{One-sFE.mpk} \leftarrow \text{One-sFE.Setup}(1^λ; r_{\text{Setup}})$

$\text{One-sFE.Enc.st} \leftarrow \text{One-sFE.EncSetup(One-sFE.mpk, r_{\text{EncSetup}})}$

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

$\text{PRF2.\text{K}} \leftarrow \text{PRF2.Setup}(1^λ; r_{\text{PRF2}})$

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

$\text{FPFE.sk}_H \leftarrow \text{FPFE.KeyGen(\text{FPFE.mpk}, H; r_{\text{H}})}$.
Then, by correctness of \( \text{FPFE} \), except with negligible probability, for all \( i \in [n] \),

\[
\text{FPFE}\text{.Dec}(\text{FPFE}\text{.sk}_H, \text{FPFE}\text{.ct}_i) = H[\text{One}\text{-sFE}.\text{msk}, \text{One}\text{-sFE}.\text{Enc}\text{.st}, \text{PRF}2.K](i, t_i, x_i, 0^{\ell_x}, 0^{\ell\text{One}\text{-sFE}\text{.ct}})
\]

\[
= \text{One}\text{-sFE}\text{.Enc}(\text{One}\text{-sFE}.\text{msk}, \text{One}\text{-sFE}.\text{Enc}\text{.st}, i, x_i; \text{PRF}2.\text{Eval}(\text{PRF}2.K, t_i))
\]

\[
= \text{One}\text{-sFE}.\text{ct}_i
\]

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

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

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

\[
\text{sFE}\text{.Dec}(SK_f, \text{Dec}\text{.ST}_1, 1, CT_1) = \text{sFE}\text{.Dec}(\text{FE}\text{.sk}_G, \perp, 1, (\text{FE}.ct, \text{FPFE}.ct_1))
\]

\[
= (y_1, \text{Dec}\text{.ST}_2 = (\text{One}\text{-sFE}\text{.sk}_f, \text{FPFE}\text{.sk}_H, \text{One}\text{-sFE}\text{.Dec}\text{.st}_2))
\]

For \( i > 1 \), using the values defined above,

\[
\text{sFE}\text{.Dec}(SK_f, \text{Dec}\text{.ST}_i, i, CT_i) = \text{sFE}\text{.Dec}(\text{FE}\text{.sk}_G, (\text{One}\text{-sFE}\text{.sk}_f, \text{FPFE}\text{.sk}_H, \text{One}\text{-sFE}\text{.Dec}\text{.st}_i), i, \text{FPFE}.ct_i)
\]

\[
= (y_i, \text{Dec}\text{.ST}_{i+1} = (\text{One}\text{-sFE}\text{.sk}_f, \text{FPFE}\text{.sk}_H, \text{One}\text{-sFE}\text{.Dec}\text{.st}_{i+1}))
\]

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.

- **Hybrid** 0<sup>A</sup>: 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 \( x_w = x_w(0), x_w(1), \ldots, x_w(n) \) with an encryption of stream \( x_w^{(1)} = x_w(1), x_w(1), \ldots, x_w(n) \).

  - **Hybrid** 0<sup>A</sup><sub>1,0</sub>: For stream identities \( \text{id} < w \), we encrypt \( x_{\text{id}}^{(1)} \) instead of \( x_{\text{id}}^{(0)} \). For \( w = 1 \), this hybrid is identical to **Hybrid** 0<sup>A</sup>.

  - **Hybrid** 0<sup>A</sup><sub>1,1</sub>: For each function key \( \text{SK}_f = \text{FE}.sk_{G_f} \), we hardwire the output of \( G_f \) on stream \( w \) into the ciphertext \( c_j \) embedded within \( G_f \). Indistinguishability follows from the pseudorandom ciphertext property of SKE.

  - **Hybrid** 0<sup>A</sup><sub>1,2</sub>: We swap ciphertext \( \text{FE}.ct_w \) from an encryption of \( (\text{FPFE}.msk_w, \text{PRF}.K_w, 0, 0^{\text{SKE}.sk}) \) to an encryption of \( (0^{\text{FPFE}.msk}, 0^{\text{PRF}.K}, 1, \text{SKE}.sk) \). Now, when decrypting \( \text{FE}.ct_w \) with \( \text{FE}.sk_{G_f} \), we will invoke the \( \alpha = 1 \) branch of \( G_f \), 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 \( (\text{FPFE}.msk_w, \text{PRF}.K_w) \) from the hybrid.

  - **Hybrid** 0<sup>A</sup><sub>1,3</sub>: We replace the values generated by \( \text{PRF}.K_w \) with true randomness. This includes the randomness used to generate the One-sFE, PRF2, and FPFE keys for stream \( w \). Indistinguishability follows from the security of PRF.

  - Within each ciphertext FPFE.ct<sub>i</sub> for stream \( w \), we encrypt both \( x_w^{(0)} \) and \( x_w^{(1)} \). Then, for each \( k \in [Q] \), we proceed through a sequence of hybrids to change \( \text{FPFE}.sk_{H_{k,w}} \) from a function key which decrypts using \( x_w^{(0)} \) to a function key which decrypts using \( x_w^{(1)} \).

    * **Hybrid** 0<sup>A</sup><sub>1,4,k,0</sub>: Within each ciphertext FPFE.ct<sub>i</sub> for stream \( w \), we additionally encrypt \( x_{w,i}^{(1)} \) and \( v_{k,w,i} = \text{FPFE}.\text{Dec}(\text{FPFE}.sk_{H_{k,w}}, \text{FPFE}.ct_{i}) \). For \( j < k \), we set function \( H_{j,w} \) to a function which computes its output using \( x_{w,j}^{(1)} \). 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 **Hybrid** 0<sup>A</sup><sub>1,4,3</sub> by the security of FPFE.

    * **Hybrid** 0<sup>A</sup><sub>1,4,k,1</sub>: We exchange the randomness generated by \( \text{PRF2}.K_w \) with true randomness. This randomness is used to compute \( v_{k,w,i} \). Indistinguishability follows from the security of PRF2.

    * **Hybrid** 0<sup>A</sup><sub>1,4,k,2</sub>: We invoke the security of One-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 One-sFE.

    * **Hybrid** 0<sup>A</sup><sub>1,4,k,3</sub>: We revert back to using \( \text{PRF2}.K_w \) to compute the randomness needed for determining \( v_{k,w,i} \). Indistinguishability follows from the security of PRF2.

    * **Hybrid** 0<sup>A</sup><sub>1,4,k,4</sub>: 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 Hybrid\textsuperscript{A}_{1,w,4,k,4} and Hybrid\textsuperscript{A}_{1,w,4,k+1,0} follows by the security of FPFE.

- **Hybrid\textsuperscript{A}_{1,w,5}:** This is identical to Hybrid\textsuperscript{A}_{1,w,4,Q,4}. Observe that every FPFE.sk\textsubscript{H,k,w} now decrypts using \(x_{w,i}^{(1)}\) instead of \(x_{w,i}^{(0)}\).

- **Hybrid\textsuperscript{A}_{1,w,6}:** Within each ciphertext FPFE.ct\textsubscript{w,i} for stream identity \(w\), we encrypt \(x_{w,i}^{(1)}\), but no longer encrypt either \(x_{w,i}^{(0)}\) or \(v_{k,w,i}\). We also change each FPFE.sk\textsubscript{H,k,w} back to its original value. Since we now only encrypt \(x_{w}^{(1)}\), we will continue to compute using stream \(x_{w}^{(1)}\). Indistinguishability follows from the security of FPFE.

- **Hybrid\textsuperscript{A}_{1,w,7}:** We revert back to using PRF.K\textsubscript{w} to generate the randomness needed for computing the One-sFE, PRF2, FPFE keys for stream \(w\). Indistinguishability follows from the security of PRF.

- **Hybrid\textsuperscript{A}_{1,w,8}:** We change FE.ct\textsubscript{w} back to its original value. Indistinguishability follows from the security of FE.

- **Hybrid\textsuperscript{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 Hybrid\textsuperscript{A}_{1,w+1,0}.

- **Hybrid\textsuperscript{A}_{2}:** This is the real world experiment with \(b = 1\). This hybrid is identical to Hybrid\textsuperscript{A}_{1,Bound,A,9} where Bound\textsubscript{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 $id = i$ 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 $id = 1$.

**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 $f(x^{(0)}_{id}) \neq f(x^{(1)}_{id})$ for some function query $f$ submitted by the adversary where $\{(x^{(0)}_{id,i}, x^{(1)}_{id,i})\}_{i \in [t]}$ are the message queries submitted so far under some stream identity $id$, $x^{(0)}_{id} = x^{(0)}_{id,1} \ldots x^{(0)}_{id,t}$, and $x^{(1)}_{id} = x^{(1)}_{id,1} \ldots x^{(1)}_{id,t}$). 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 One-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 $P/\text{Poly}$, and (2) a single-key, single-ciphertext, adaptive-IND-secure, secret-key sFE scheme for $P/\text{Poly}$, there exists a $Q$-bounded adaptive-IND-secure, public-key (resp. secret-key) sFE scheme for $P/\text{Poly}$.

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 One-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.
Hybrid$_0^4(1^\lambda)$: 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 Bound$_A$ stream identities $id$ during the hybrid where Bound$_A$ is a bound on the runtime of $A$.

1. **Parameters:** The adversary $A$ receives security parameter $1^\lambda$, and outputs a function size $1^\ell_F$, a state size $1^\ell_S$, an input size $1^\ell_X$, and an output size $1^\ell_Y$.

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

3. **Encryption Setup:** For $id \in [\text{Bound}_A]$ where Bound$_A$ is a bound on the runtime of $A$.
   (a) $PRF.K_id \leftarrow PRF.$Setup$(1^\lambda)$.
   (b) $FPFE.msk_id \leftarrow FPFE.$Setup$(1^\lambda)$.
   (c) $FE.ct_id \leftarrow FE.$Enc$(FE.ek, (FPFE.msk_id, PRF.K_id, 0, 0^{\ell_{\text{SKE.ct}}}))$.

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 $f_j \in F[\ell_F, \ell_S, \ell_X, \ell_Y]$ made by the adversary:
      i. $s_j \leftarrow \{0, 1\}^\lambda$.
      ii. $c_j \leftarrow \{0, 1\}^{\ell_{\text{SKE.ck}}}$.
      iii. Let $G_j = G[f_j, s_j, c_j]$ as defined in Figure 9 (page 50).
      iv. $FE.sk_{G_j} \leftarrow FE.$KeyGen$(FE.msk, G_j)$.
      v. Send $SK_{f_j} = FE.sk_{G_j}$ to the adversary.
   (b) **Message Query:** For the $i^{th}$ message query made to stream identity $id$, $A$ outputs a message pair $(x_{id,i}^{(0)}, x_{id,i}^{(1)})$ where $x_{id,i}^{(0)}, x_{id,i}^{(1)} \in \{0, 1\}^{\ell_X}$.
      i. $t_{id,i} \leftarrow \{0, 1\}^\lambda$.
      ii. $FPFE.ct_{id,i} \leftarrow FPFE.$Enc$(FPFE.msk_id, (i, t_{id,i}, x_{id,i}^{(0)}, 0^{\ell_X}, 0^{\ell_{\text{One-Time-ct}}})))$.
      iii. If $i = 1$, set $CT_{id,1} = (FE.ct_id, FPFE.ct_{id,1})$. Else, set $CT_{id,i} = FPFE.ct_{id,i}$.
      iv. Send $CT_{id,i}$ to the adversary.

6. **Experiment Outcome:** $A$ outputs a bit $b'$ which is the output of the experiment.
**Hybrid**\(^4\)\(_{1,w,0}(1^\lambda)\): For stream identities \(id < w\), we encrypt \(x_{id}^{(1)}\) instead of \(x_{id}^{(0)}\).

This is the same as **Hybrid**\(^4\)\(_0\) except that we change the following steps:

5b. **Message Query**: For the \(i\)th message query made to stream identity \(id\), \(A\) outputs a message pair \((x_{id,i}^{(0)}, x_{id,i}^{(1)})\) where \(x_{id,i}^{(0)}, x_{id,i}^{(1)} \in \{0, 1\}^\ell_X\).

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

   (b) If \(id < w\), \(\text{FPFE.ct}_{id,i} \leftarrow \text{FPFE.Enc(\text{FPFE.msk}_{id}, (i, t_{id,i}, x_{id,i}^{(1)}, 0^\ell_X, 0^\ell_{\text{One}-\text{SEFE}}, ct))}\).

   (c) If \(id \geq w\), \(\text{FPFE.ct}_{id,i} \leftarrow \text{FPFE.Enc(\text{FPFE.msk}_{id}, (i, t_{id,i}, x_{id,i}^{(0)}, 0^\ell_X, 0^\ell_{\text{One}-\text{SEFE}}, ct))}\).

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

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

**Lemma 5.9.** For all adversaries \(A\),

\[
\left| \Pr[\text{Hybrid}_0^4(1^\lambda) = 1] - \Pr[\text{Hybrid}_{1,1,0}^4(1^\lambda) = 1] \right| = 0
\]

**Proof.** The hybrids are identical since we have assumed that each stream identity \(id \geq 1\). (See Remark 5.6.) \(\square\)
\textbf{Hybrid}^{4}_{1,w,1}(1^\lambda): For each \( j \), we hardcode into \( c_j \) the values

\[
(\text{One-sFE.sk}_{f_j,w}, \text{FPFE.sk}_{H_j,w}) = G_j (\text{FPFE.msk}_w, \text{PRF}.K_w, 0, 0^{\text{SKE.w}})
\]

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 \textbf{Hybrid}^{4}_{1,w,0} except that we change the following steps:

2. **Setup:**

- \textit{Secret-Key Setting}:
  
  (a) \( \text{SKE.sk} \leftarrow \text{SKE.Setup}(1^\lambda) \).
  
  (b) \( \text{FE.msk} \leftarrow \text{FE.Setup}(1^\lambda) \).
  
  (c) \( \text{FE.ek} = \text{FE.msk} \).

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

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

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

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

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

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

(e) \( \text{PRF2}.K_{j, w} \leftarrow \text{PRF2.Setup}(1^\lambda; \ell_{\text{PRF2}}, j, w) \).

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

(g) \( \text{FPFE.sk}_{H_{j, w}} \leftarrow \text{FPFE.KeyGen}(\text{FPFE.msk}_{w}, H_{j, w}; \ell_{H}, j, w) \).

5a. **Function Query:** For the \( j^{th} \) function query \( f_j \in F[\ell_{\text{F}}, \ell_{\text{G}}, \ell_{\text{X}}, \ell_{\text{Y}}] \) made by the adversary:

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

(b) \( \text{One-sFE.sk}_{f_j, w} \leftarrow \text{One-sFE.KeyGen}(\text{One-sFE.msk}_{j, w}, f_j; \ell_{\text{KeyGen}}, j, w) \).

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

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

(e) \( \text{FE.sk}_{G_j} \leftarrow \text{FE.KeyGen}(\text{FE.msk}, G_j) \).

(f) Send \( \text{SK}_{f_j} = \text{FE.sk}_{G_j} \) to the adversary.

\textbf{Lemma 5.10.} If SKE has pseudorandom ciphertexts, then for all PPT adversaries \( A \) and all \( w \in [\text{Bound}_A] \),

\[
\left| \Pr[\text{Hybrid}^{4}_{1,w,0}(1^\lambda) = 1] - \Pr[\text{Hybrid}^{4}_{1,w,1}(1^\lambda) = 1] \right| \leq \text{negl}(\lambda)
\]
Proof. Suppose for sake of contradiction that there exists a PPT adversary $A$ and a $w \in \text{Bound}_A$ such that

$$\left| \Pr[\text{Hybrid}_{A,0}(1^\lambda) = 1] - \Pr[\text{Hybrid}_{A,1}(1^\lambda) = 1] \right| > \text{negl}(\lambda)$$

(6)

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

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

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

After $A$ is done making queries, $A$ outputs $b'$ which $B$ also outputs. If the experiment for $A$ aborts for any reason, $B$ instead outputs 0. Observe that if every $c_{j,w}$ is an independent uniform random value, then $B$ exactly emulates $\text{Hybrid}_{A,0}$, and if each $c_{j,w}$ is an encryption of $m_{j,w}$, then $B$ emulates $\text{Hybrid}_{A,1}$. Additionally, $B$ does not need to know $\text{SKE.sk}$ to carry out this experiment. Thus, by Equation 6, this means that $B$ breaks the pseudorandom ciphertext property of SKE as $B$ can distinguish between receiving random values and valid ciphertexts with non-negligible probability. \qed
Hybrid$^A_{1,w,2}(1^\lambda)$: We change the message encrypted in $\text{FE.ct}_w$ so that we use the $\alpha = 1$ branch of every $G_j$. This allows us to remove $\text{FPFE.msk}_w$ and $\text{PRF.K}_w$ from $\text{FE.ct}_w$.

This is the same as Hybrid$^A_{1,w,1}$ except that we change the following steps:

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

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

$$\Pr[\text{Hybrid}^A_{1,w,1}(1^\lambda) = 1] - \Pr[\text{Hybrid}^A_{1,w,2}(1^\lambda) = 1] \leq \text{negl}(\lambda)$$

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

$$\Pr[\text{Hybrid}^A_{1,w,1}(1^\lambda) = 1] - \Pr[\text{Hybrid}^A_{1,w,2}(1^\lambda) = 1] > \text{negl}(\lambda) \quad (7)$$

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

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

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

- For each function query $f_j$ output by $A$, $B$ does the following: $B$ computes $G_j = G[f_j, s_j, c_{j,w}]$ as in step 5a of Hybrid$^A_{1,w,2}$. $B$ sends function query $G_j$ to its FE challenger and receives a function key $\text{FE.sk}_{G_j}$. This is a valid function query since for all $j \in [Q],
\begin{align*}
G[f_j, s_j, c_{j,w}] &\leftarrow (\text{FPFE.msk}_w, \text{PRF.K}_w, 0^{\text{SKE.sk}}) = G[f_j, s_j, c_{j,w}](0^{\text{FPFE.msk}}, 0^{\text{PRF.K}}, 1, \text{SKE.sk})
\end{align*}$

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

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

After $A$ is done making queries, $A$ outputs $b'$ which $B$ also outputs. If the experiment for $A$ aborts for any reason, $B$ instead outputs 0. Observe that if $B$ received $\text{FE}$ ciphertexts for the first message in each challenge pair (i.e. either $m_{0,w}$ or $m_{id}$), then $B$ exactly emulates Hybrid$^A_{1,w,1}$, and if $B$ received $\text{FE}$ ciphertexts for the second message in each challenge pair (i.e. either $m_{1,w}$ or $m_{id}$),
then $\mathcal{B}$ emulates $\text{Hybrid}^4_{1,w,2}$. Additionally, $\mathcal{B}$ does not need to know $\text{FE.ms}k$ 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 $\text{FE}$ as $\mathcal{B}$ can distinguish between the two security games with non-negligible probability. 

Hybrid\textsuperscript{4}\textsubscript{1,w,3}: We exchange the randomness generated by PRF.\textsubscript{K,w} with true randomness. This is the same as Hybrid\textsuperscript{4}\textsubscript{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) \( (\mathsf{Setup}_{j,w}, \mathsf{EncSetup}_{j,w}, \mathsf{KeyGen}_{j,w}, \mathsf{PRF2}_{j,w}, \mathsf{H}_{j,w}) \leftarrow \mathsf{PRF.Eval(Prf.K.w, s_j)} \).
   
   (c) One-sFE.\mathsf{msk}_{j,w} \leftarrow \text{One-sFE.Setup}(1^\lambda;\mathsf{Setup}_{j,w})
   
   (d) One-sFE.\mathsf{Enc.st}_{j,w} \leftarrow \text{One-sFE.EncSetup}(\text{One-sFE.\mathsf{msk}_{j,w}};\mathsf{EncSetup}_{j,w})
   
   (e) \( \mathsf{PRF2.K}_{j,w} \leftarrow \text{PRF2.Setup}(1^\lambda; \mathsf{PRF2.K.w}) \).
   
   (f) Let \( \mathsf{H}_{j,w} = H[\text{One-sFE.\mathsf{msk}_{j,w}}, \text{One-sFE.\mathsf{Enc.st}_{j,w}}, \mathsf{PRF2.K}_{j,w}] \) as defined in Figure 10 (page 50).
   
   (g) \( \mathsf{FPFE.sk}_{H,j,w} \leftarrow \text{FPFE.KeyGen(\mathsf{FPFE.\mathsf{msk}_{w}}, \mathsf{H}_{j,w};\mathsf{H}_{j,w})} \).

5a. **Function Query**: For the \( j \text{'th} \) function query \( f_j \in \mathcal{F}[\ell_x, \ell_y, \ell_x, \ell_y] \) made by the adversary:
   
   (a) One-sFE.\mathsf{sk}_{f_j} \leftarrow \text{One-sFE.KeyGen(One-sFE.\mathsf{msk}_{j,w}, f_j; \mathsf{KeyGen}_{j,w})}
   
   (b) \( c_{j,w} \leftarrow \mathsf{SKE.Enc(\mathsf{SKE.sk};(\text{One-sFE.\mathsf{sk}_{f_j}}, \mathsf{FPFE.\mathsf{sk}_{H,j,w}}))} \).
   
   (c) Let \( G_j = G[f_j, s_j, c_{j,w}] \) as defined in Figure 9 (page 50).
   
   (d) \( \mathsf{FE.sk}_{G_j} \leftarrow \mathsf{FE.KeyGen(\mathsf{FE.\mathsf{msk}}, G_j)} \).
   
   (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 \( A \) and all \( w \in [\text{Bound}_A] \),

\[
\Pr[\text{Hybrid}^4_{1,w,2}(1^\lambda) = 1] - \Pr[\text{Hybrid}^4_{1,w,3}(1^\lambda) = 1] \leq \text{negl}(\lambda)
\]

**Proof.** Suppose for sake of contradiction that there exists a PPT adversary \( A \) and a \( w \in [\text{Bound}_A] \) such that

\[
\Pr[\text{Hybrid}^4_{1,w,2}(1^\lambda) = 1] - \Pr[\text{Hybrid}^4_{1,w,3}(1^\lambda) = 1] > \text{negl}(\lambda)
\]

(8)

We build a PPT adversary \( B \) that breaks the security of PRF. \( B \) first runs steps 1-3 as in Hybrid\textsuperscript{4}\textsubscript{1,w,3} except that \( B \) does not compute PRF.\textsubscript{K.w}.

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

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

At the final step, \( A \) outputs \( b' \) which \( B \) also outputs. If the experiment for \( A \) aborts for any reason, \( B \) instead outputs 0. Observe that if \( B \)’s PRF oracle was a uniform random function \( R \), then \( B \) exactly emulates Hybrid\textsuperscript{4}\textsubscript{1,w,2}, and if \( B \)’s PRF oracle was PRF.Eval(Prf.K.w, \cdot) for some PRF key PRF.K.w, then \( B \) emulates Hybrid\textsuperscript{4}\textsubscript{1,w,3}. Additionally, \( B \) does not need to know PRF.K.w to carry out this experiment. Thus, by Equation 8, this means that \( B \) breaks the security of PRF as \( B \) can distinguish between a random function and PRF evaluations.

\( \square \)
Hybrid$_{1, w, 4, k, 0}(1^\lambda)$: 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 function one at a time.

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

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 $FPFE.ct_{w, i}$ to $v_{k, w, i}$ which we compute as the output of the original $H_{k, w}$ on $(i, t_{w, i}, x_{w, i}^{(0)}, 0^x, 0^{\text{One-sFE.a}})$. These changes allow us to remove $(\text{One-sFE.msk}_{k, w}, \text{One-sFE.Enc.st}_{k, w}, \text{PRF2.K}_{k, w})$ from $FPFE.sk_{H_{k, w}}$.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter $1^\lambda$, and outputs a function size $1^{\ell_f}$, a state size $1^{\ell_s}$, an input size $1^{\ell_x}$, and an output size $1^{\ell_y}$.

2. **Setup**:

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

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

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

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

   - (a) $s_j \leftarrow \{0, 1\}^\lambda$.
   - (b) $\text{One-sFE.msk}_{j, w} \leftarrow \text{One-sFE.Setup}(1^\lambda)$.
   - (c) $\text{One-sFE.Enc.st}_{j, w} \leftarrow \text{One-sFE.EncSetup}(\text{One-sFE.msk}_{j, w})$.
   - (d) $\text{PRF2.K}_{j, w} \leftarrow \text{PRF2.Setup}(1^\lambda)$.
   - (e) If $j < k$, let $H_{j, w} = H'[\text{One-sFE.msk}_{j, w}, \text{One-sFE.Enc.st}_{j, w}, \text{PRF2.K}_{j, w}]$ 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 $H_{j, w} = H[\text{One-sFE.msk}_{j, w}, \text{One-sFE.Enc.st}_{j, w}, \text{PRF2.K}_{j, w}]$ as defined in Figure 10 (page 50).
   - (h) $\text{FPFE.sk}_{H_{j, w}} \leftarrow \text{FPFE.KeyGen}(\text{FPFE.msk}_{w}, H_{j, w})$.
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 $f_j \in \mathcal{F}[\ell_F, \ell_S, \ell_X, \ell_Y]$ made by the adversary:

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

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

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

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

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

(b) **Message Query:** For the $i^{th}$ message query made to stream identity $\text{id}$, $\mathcal{A}$ outputs a message pair $(x_{i,d,i}^{(0)}, x_{i,d,i}^{(1)})$ where $x_{i,d,i}^{(0)}, x_{i,d,i}^{(1)} \in \{0, 1\}^{\ell_x}$.

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

ii. If $\text{id} < w$, $\text{FPFE.ct}_{i,d,i} \leftarrow \text{FPFE.Enc}(\text{FPFE.msk}_{\text{id}}, (i, t_{i,d,i}, x_{i,d,i}^{(1)}, q^{\text{One-sFE}^*}))$

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

   A. $r_{k,w,i} \leftarrow \text{PRF2.Eval}(\text{PRF2.K}_{k,w}, t_{w,i})$

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

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

iv. If $\text{id} > w$, $\text{FPFE.ct}_{i,d,i} \leftarrow \text{FPFE.Enc}(\text{FPFE.msk}_{\text{id}}, (i, t_{i,d,i}, x_{i,d,i}^{(0)}, q^{\text{One-sFE}^*}))$

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

vi. Send $\text{CT}_{i,d,i}$ 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 $w \in [\text{Bound}_A]$,

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

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

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

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 $\text{Hybrid}_{1,w,3}^A$ except that $\mathcal{B}$ does not compute $\text{FPFE.msk}_w$.

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

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

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

- For each function query $f_j$ output by $\mathcal{A}$, $\mathcal{B}$ computes $\text{SK}_{f_j}$ as in step 5a of $\text{Hybrid}_{1,w,3}^A$, and sends $\text{SK}_{f_j}$ to $\mathcal{A}$.
For each message query \((x_{id,i}^{(0)}, x_{id,i}^{(1)})\) output by \(A:\)

If \(id \neq w\), \(B\) computes \(CT_{id,i}\) as in step 5a of \(\text{Hybrid}^{A}_{1,w,3}\), and sends \(CT_{id,i}\) to \(A\).

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

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

If \(i = 1\), \(B\) sets \(CT_{w,1} = (\text{FE}.ct_{w}, \text{FPFE}.ct_{w,1})\). Else, \(B\) sets \(CT_{w,i} = \text{FPFE}.ct_{w,i}\). \(B\) sends \(CT_{w,i}\) to \(A\).

After \(A\) is done making queries, \(A\) outputs \(b'\) which \(B\) also outputs. If the experiment for \(A\) aborts for any reason, \(B\) instead outputs 0. Observe that if \(B\) received only ciphertexts and function keys for the first message or function of each of its challenge pairs, then \(B\) exactly emulates \(\text{Hybrid}^{A}_{1,w,3}\), and if \(B\) received only ciphertexts and function keys for the second message or function of each of its challenge pairs, then \(B\) emulates \(\text{Hybrid}^{A}_{1,w,4,1,0}\). Additionally, \(B\) does not need to know \(\text{FPFE}.msk_w\) to carry out this experiment and makes only \(Q\) function queries. Thus, by Equation 9, this means that \(B\) breaks the \(Q\)-bounded, function-private, function-selective-IND-security of \(\text{FPFE}\) as \(B\) can distinguish between the two security games with non-negligible probability. \(\Box\)
Hybrid$_{1,w,4,k,1}(1^\lambda)$: We exchange the randomness generated by PRF2.$K_{k,w}$ with true randomness. This is the same as Hybrid$_{1,w,4,k,0}(1^\lambda)$ except that we change the following steps:

5b. Message Query: For the $i^{th}$ message query made to stream identity $id$, $A$ outputs a message pair $(x_{id,i}^{(0)}, x_{id,i}^{(1)})$ where $x_{id,i}^{(0)}, x_{id,i}^{(1)} \in \{0,1\}^\ell$.

   (a) $t_{id,i} \leftarrow \{0,1\}^\lambda$.
   (b) If $id < w$, $FPFE.ct_{id,i} \leftarrow FPFE.Enc(FPFE.msk_{id}, (i, t_{id,i}, x_{id,i}^{(1)}, 0^{\ell - \lambda}, 0^{\ell \cdot \text{One-sFE.ct}}))$.
   (c) If $id = w$,
      i. $\ast_{k,w,i} \leftarrow \text{PRF2.Eval}($PRF2.$K_{k,w}, t_{w,i})$.
      ii. $v_{k,w,i} \leftarrow \text{One-sFE.Enc}(\text{One-sFE.msk}_{k,w}, \text{One-sFE.Enc.st}_{k,w}, i, x_{w,i}^{(0)})$.
      iii. $FPFE.ct_{w,i} \leftarrow FPFE.Enc(FPFE.msk_{w}, (i, t_{w,i}, x_{w,i}^{(0)}, v_{k,w,i}))$.
   (d) If $id > w$, $FPFE.ct_{id,i} \leftarrow FPFE.Enc(FPFE.msk_{id}, (i, t_{id,i}, x_{id,i}^{(0)}, 0^{\ell - \lambda}, 0^{\ell \cdot \text{One-sFE.ct}}))$.
   (e) If $i = 1$, set $CT_{id,1} = (\text{FE.ct}_{id}, FPFE.ct_{id,1})$. Else, set $CT_{id,i} = FPFE.ct_{id,i}$.
   (f) Send $CT_{id,i}$ to the adversary.

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

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

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

$$\left| \Pr[\text{Hybrid}_{1,w,4,k,0}(1^\lambda) = 1] - \Pr[\text{Hybrid}_{1,w,4,k,1}(1^\lambda) = 1] \right| > \text{negl}(\lambda) \quad (10)$$

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

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

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

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

66
Hybrid\textsuperscript{4}\textsubscript{1, w, 4, k, 2}(1^\lambda): We now invoke the security of One-sFE to change each \(v_{k, w, i}\) from an encryption of \(x_{w, i}^{(0)}\) to an encryption of \(x_{w, i}^{(1)}\).

This is the same as Hybrid\textsuperscript{4}\textsubscript{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 \(id\), \(A\) outputs a message pair \((x_{id, i}^{(0)}, x_{id, i}^{(1)})\) where \(x_{id, i}^{(0)}, x_{id, i}^{(1)} \in \{0, 1\}^\ell\).

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

(b) If \(id < w\), \(FPFE.ct_{id, i} \leftarrow FPFE.Enc(FPFE.msk_{id}, (i, t_{id, i}, x_{id, i}^{(1)}, 0^\lambda, 0^{\text{One-sFE. ct}}))\).

(c) If \(id = w\),

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

ii. \(FPFE.ct_{w, i} \leftarrow FPFE.Enc(FPFE.msk_{w}, (i, t_{w, i}, x_{w, i}^{(0)}, x_{w, i}^{(1)}, v_{k, w, i}))\).

(d) If \(id > w\), \(FPFE.ct_{id, i} \leftarrow FPFE.Enc(FPFE.msk_{id}, (i, t_{id, i}, x_{id, i}^{(0)}, 0^\lambda, 0^{\text{One-sFE. ct}}))\).

(e) If \(i = 1\), set \(CT_{id, 1} = (\text{FE.ct}_{id, FPFE.ct_{id, 1}})\). Else, set \(CT_{id, i} = FPFE.ct_{id, i}\).

(f) Send \(CT_{id, i}\) to the adversary.

\textbf{Lemma 5.15.} If One-sFE is single-key, single-ciphertext, function-selective-IND-secure, then for all PPT adversaries \(A\), all \(w \in [\text{Bound}_A]\), and all \(k \in [Q]\),

\[
\Pr[\text{Hybrid}\textsuperscript{4}\textsubscript{1, w, 4, k, 1}(1^\lambda) = 1] - \Pr[\text{Hybrid}\textsuperscript{4}\textsubscript{1, w, 4, k, 2}(1^\lambda) = 1] \leq \text{negl}(\lambda)
\]

\textbf{Proof.} Suppose for sake of contradiction that there exists a PPT adversary \(A\), \(w \in [\text{Bound}_A]\), and \(k \in [Q]\) such that

\[
\Pr[\text{Hybrid}\textsuperscript{4}\textsubscript{1, w, 4, k, 1}(1^\lambda) = 1] - \Pr[\text{Hybrid}\textsuperscript{4}\textsubscript{1, w, 4, k, 2}(1^\lambda) = 1] > \text{negl}(\lambda) \quad (11)
\]

We build a PPT adversary \(B\) that breaks the single-key, single-ciphertext, function-selective-IND-security of One-sFE. \(B\) first runs steps as in 1-4 of Hybrid\textsuperscript{4}\textsubscript{1, w, 4, k, 2} except that \(B\) does not compute \((\text{One-sFE.msk}_{k, w}, \text{One-sFE.Enc.st}_{k, w})\) 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 \(A\):
  - If \(j \neq k\), \(B\) computes \(SK_{f_j}\) as in step 5a of Hybrid\textsuperscript{4}\textsubscript{1, w, 4, k, 2}, and sends \(SK_{f_j}\) to \(A\).
  - If \(j = k\), \(B\) sends \(f_k\) to its One-sFE challenger and receives a function key \(\text{One-sFE.sk}_{f_k}\). \(B\) computes \(c_{k, w} \leftarrow \text{SKE.Enc}(\text{SKE.sk}, (\text{One-sFE.sk}_{f_k}, \text{FPFE.sk}_{H_{k, w}}))\) and \(\text{FE.sk}_{G_k} \leftarrow \text{FE.KeyGen}(\text{FE.msk}, G_k)\) for \(G_k = G[f_k, s_k, c_{k, w}]\). \(B\) sends \(\text{SK}_{f_k} = \text{FE.sk}_{G_k}\) to \(A\).

- For each message query \((x_{id, i}^{(0)}, x_{id, i}^{(1)})\) output by \(A\):
  - If \(id \neq w\), \(B\) computes \(CT_{id, i}\) as in step 5b of Hybrid\textsuperscript{4}\textsubscript{1, w, 4, k, 2}, and sends \(CT_{id, i}\) to \(A\).
  - If \(id = w\), \(B\) sends challenge message pair \((x_{w, i}^{(0)}, x_{w, i}^{(1)})\) to its One-sFE challenger and receives \(\text{One-sFE.ct}_{w, i}\). This is a valid message query since \(A\) is restricted to function and message queries that satisfy
    \[
    f_k(x_{w}^{(0)}) = f_k(x_{w}^{(1)})
    \]
    \(B\) then computes \(t_{w, i} \leftarrow \{0, 1\}^\lambda\) and \(\text{FPFE.ct}_{w, i} \leftarrow \text{FPFE.Enc}(\text{FPFE.msk}_{w}, (i, t_{w, i}, x_{w, i}^{(0)}, x_{w, i}^{(1)}, \text{One-sFE.ct}_{w, i}))\).
  - If \(i = 1\), \(B\) sets \(CT_{w, 1} = (\text{FE.ct}_{w, \text{FPFE.ct}_{w, 1}})\). Else, \(B\) sets \(CT_{w, i} = \text{FPFE.ct}_{w, i}\). \(B\) sends \(CT_{w, i}\) to \(A\).
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 One-sFE challenger, then $\mathcal{B}$ exactly emulates $\text{Hybrid}^A_{1,w,4,k,1}$, and if $\mathcal{B}$ received ciphertexts for $x_w^{(1)}$, then $\mathcal{B}$ emulates $\text{Hybrid}^A_{1,w,4,k,2}$. Additionally, $\mathcal{B}$ does not need to know $(\text{One-sFE}.\text{msk}_{k,w}, \text{One-sFE}.\text{Enc}.\text{st}_{k,w})$ to carry out this experiment and makes only one function query followed by one message query to its One-sFE challenger. Thus, by Equation 11, this means that $\mathcal{B}$ breaks the single-key, single-ciphertext, function-selective-IND-security of One-sFE as $\mathcal{B}$ can distinguish between the two security games with non-negligible probability.
**Hybrid**<sub>1,w,4,k,3</sub>(1<sup>λ</sup>): We revert back to using PRF.<sub>2</sub>.<sub>K,k,w</sub> to compute the randomness needed for determining each v<sub>k,w,i</sub>. This is the same as **Hybrid**<sub>1,w,4,k,2</sub> except that we change the following steps:

5b. **Message Query:** For the i<sup>th</sup> message query made to stream identity id, A outputs a message pair (x<sub>(0)id,i</sub>, x<sub>(1)id,i</sub>) where x<sub>(0)id,i</sub>, x<sub>(1)id,i</sub> ∈ {0, 1}<sup>ℓ</sup>.

(a) \( t_{id,i} \leftarrow \{0, 1\}^{λ} \).
(b) If id < w, FPFE.<sub>ct</sub><sub>id,i</sub> ← FPFE.Enc(FPFE.<sub>msk</sub><sub>id</sub>, (i, t<sub>id,i</sub>, x<sub>(1)id,i</sub>, x<sub>(0)id,i</sub>, 0<sup>ℓ</sup>), 0<sup>ℓ</sup>One-sFE.<sub>ct</sub>).
(c) If id = w,
   i. \( r_{k,w,i} \leftarrow \text{PRF2.Eval(PRF2}.K_{k,w}, t_{w,i}) \).
   ii. \( v_{k,w,i} \leftarrow \text{One-sFE.Enc(One-sFE.<sub>msk</sub>}_{k,w}, \text{One-sFE.Enc.st}_{k,w}, i, x_{w,i}^{(1)}, r_{k,w,i}) \).
   iii. \( \text{FPFE.<sub>ct</sub>}_{w,i} \leftarrow \text{FPFE.Enc(FPFE.<sub>msk</sub>}_{w}, (i, t_{w,i}, x_{w,i}^{(0)}, x_{w,i}^{(1)}), v_{k,w,i}) \).
(d) If id > w, FPFE.<sub>ct</sub><sub>id,i</sub> ← FPFE.Enc(FPFE.<sub>msk</sub><sub>id</sub>, (i, t<sub>id,i</sub>, x<sub>(0)id,i</sub>, x<sub>(1)id,i</sub>, 0<sup>ℓ</sup>One-sFE.<sub>ct</sub>)).
(e) If i = 1, set CT<sub>id,1</sub> = (FE.<sub>ct</sub><sub>id</sub>, FPFE.<sub>ct</sub><sub>id,1</sub>). Else, set CT<sub>id,i</sub> = FPFE.<sub>ct</sub><sub>id,i</sub>.
(f) Send CT<sub>id,i</sub> to the adversary.

**Lemma 5.16.** If PRF2 is a secure PRF, then for all PPT adversaries A, all \( w \in \text{Bound}_A \), and all \( k \in \mathbb{Q} \),

\[
\left| \Pr[\text{Hybrid}_{1,w,4,k,3}^{A}(1^{λ}) = 1] - \Pr[\text{Hybrid}_{1,w,4,k,2}^{A}(1^{λ}) = 1] \right| \leq \text{negl}(λ)
\]

**Proof.** This proof is essentially the same as the proof of Lemma 5.14. \(\square\)
Hybrid$_{1,w,4,k,4}(1^\lambda)$: We change function $H_{j,w}$ from $H^*$ to $H'[\text{One-sFE}\text{-msk}_{j,w}, \text{One-sFE}\text{-Enc}\text{st}_{j,w}, \text{PRF2}\text{-K}_{j,w}]$, which operates on the second stream input given, namely $x^{(1)}_{w,i}$.

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

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

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

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

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

$$\left| \Pr[\text{Hybrid}_{1,w,4,k,3}(1^\lambda) = 1] - \Pr[\text{Hybrid}_{1,w,4,k,4}(1^\lambda) = 1] \right| > \text{negl}(\lambda) \quad (12)$$

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 Hybrid$_{1,w,4,k,4}$ except that $\mathcal{B}$ does not compute FPFE.$\text{msk}_{w,j}$.

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

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

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

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

70
For each message query \((x_{id,i}^{(0)}, x_{id,i}^{(1)})\) output by \(A\):

If \(id \neq w\), \(B\) computes \(CT_{id,i}\) as in step 5b of \(\text{Hybrid}^A_{1,w,4,k,4}\), and sends \(CT_{id,i}\) to \(A\).

If \(id = w\), \(B\) computes \((t_{w,i}, v_{k,w,i})\) as in step 5b of \(\text{Hybrid}^A_{1,w,4,k,4}\). \(B\) sends challenge message pair \((i, t_{w,i}, x_{w,i}^{(0)}, x_{w,i}^{(1)}, v_{k,w,i}), (i, t_{w,i}, x_{w,i}^{(0)}, x_{w,i}^{(1)}, v_{k,w,i})\) to its \(\text{FPFE}\) challenger and receives \(\text{FPFE} \cdot ct_{w,i}\). This is a valid message query since

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

If \(i = 1\), \(B\) sets \(CT_{w,1} = (\text{FE} \cdot ct_{w}, \text{FPFE} \cdot ct_{w,1})\). Else, \(B\) sets \(CT_{w,i} = \text{FPFE} \cdot ct_{w,i}\). \(B\) sends \(CT_{w,i}\) to \(A\).

After \(A\) is done making queries, \(A\) outputs \(b'\) which \(B\) also outputs. If the experiment for \(A\) aborts for any reason, \(B\) instead outputs 0. Observe that if \(B\) received only ciphertexts and function keys for the first message or function of each of its challenge pairs, then \(B\) exactly emulates \(\text{Hybrid}^A_{1,w,4,k,3}\), and if \(B\) received only ciphertexts and function keys for the second message or function of each of its challenge pairs, then \(B\) emulates \(\text{Hybrid}^A_{1,w,4,k,4}\). Additionally, \(B\) does not need to know \(\text{FPFE} \cdot msk_w\) to carry out this experiment and makes only \(Q\) function queries. Thus, by Equation 12, this means that \(B\) breaks the \(Q\)-bounded, function-private, function-selective-IND-security of \(\text{FPFE}\) as \(B\) can distinguish between the two security games with non-negligible probability.

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

\[
\left| \Pr[\text{Hybrid}^A_{1,w,4,k,4}(1^\lambda) = 1] - \Pr[\text{Hybrid}^A_{1,w,4,k+1,0}(1^\lambda) = 1] \right| \leq \text{negl}(\lambda)
\]

**Proof.** Suppose for sake of contradiction that there exists a PPT adversary \(A\), \(w \in \text{Bound}_A\), and \(k \in [Q - 1]\) such that

\[
\left| \Pr[\text{Hybrid}^A_{1,w,4,k,4}(1^\lambda) = 1] - \Pr[\text{Hybrid}^A_{1,w,4,k+1,0}(1^\lambda) = 1] \right| > \text{negl}(\lambda) \quad (13)
\]

We build a PPT adversary \(B\) that breaks the \(Q\)-bounded, function-private, function-selective-IND-secure of \(\text{FPFE}\). \(B\) first runs steps 1-3 of \(\text{Hybrid}^A_{1,w,4,k,4}\) except that \(B\) does not compute \(\text{FPFE} \cdot msk_w\).

For \(j \in [Q]\), \(B\) does the following: \(B\) computes \((s_j, \text{One-sFE} \cdot msk_{j,w}, \text{One-sFE} \cdot \text{Enc}.s_{j,w}, \text{PRF2}.K_{j,w})\) as in step 4 of \(\text{Hybrid}^A_{1,w,4,k,4}\). \(B\) sets \(H_{j,w}^{(0)} = H[\text{One-sFE} \cdot msk_{j,w}, \text{One-sFE} \cdot \text{Enc}.s_{j,w}, \text{PRF2}.K_{j,w}]\) and \(H_{j,w}^{(1)} = H'[\text{One-sFE} \cdot msk_{j,w}, \text{One-sFE} \cdot \text{Enc}.s_{j,w}, \text{PRF2}.K_{j,w}]\).

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

\(B\) then sends all \(Q\) challenge function pairs to its \(\text{FPFE}\) challenger and receives \(\{\text{FPFE} \cdot sk_{H_{j,w}}\}_{j \in [Q]}\).
• For each function query $f_j$ output by $A$, $B$ computes $SK_{f_j}$ as in step 5a of $Hybrid_{1,w,k,a}^A$, and sends $SK_{f_j}$ to $A$.

• For each message query $(x_{0}^{(0)}, x_{1}^{(1)})$ output by $A$:
  
  If $id \neq w$, $B$ computes $CT_{id,i}$ as in step 5b of $Hybrid_{1,w,4,k,4}^A$, and sends $CT_{id,i}$ to $A$.
  
  If $id = w$, $B$ samples $t_{w,i} \leftarrow \{0,1\}^\lambda$ and uses this to compute $v_{k,w,i}$ as in step 5b of $Hybrid_{1,w,4,k,4}^A$. $B$ sends challenge message pair $((i, t_{w,i}, x_{0}^{(0)}, x_{1}^{(1)}, v_{k,w,i}), (i, t_{w,i}, x_{0}^{(0)}, x_{1}^{(1)}, v_{k+1,w,i}))$ to its FPFE challenger and receives $FPFE_{ct_{w,i}}$. This is a valid message query since

  - For $j = k + 1$, $H_{k+1,w}^0(i, t_{w,i}, x_{0}^{(0)}, x_{1}^{(1)}, v_{k,w,i}) = H^*(i, t_{w,i}, x_{0}^{(0)}, x_{1}^{(1)}, v_{k+1,w,i})$
    
    - For $j \in [Q] \setminus \{k + 1\}$,
    
    $H_{j,w}^0(i, t_{w,i}, x_{0}^{(0)}, x_{1}^{(1)}, v_{k,w,i}) = H_{j,w}^0(i, t_{w,i}, x_{0}^{(0)}, x_{1}^{(1)}, v_{k+1,w,i})$
    
    $H_{j,w}^1(i, t_{w,i}, x_{0}^{(0)}, x_{1}^{(1)}, v_{k,w,i}) = H_{j,w}^1(i, t_{w,i}, x_{0}^{(0)}, x_{1}^{(1)}, v_{k+1,w,i})$
    
    since $H_{j,w}^0$ and $H_{j,w}^1$ ignore the last input.

  If $i = 1$, $B$ sets $CT_{w,1} = (FE_{ct_{w}}, FPFE_{ct_{w,1}})$. Else, $B$ sets $CT_{w,i} = FPFE_{ct_{w,i}}$. $B$ sends $CT_{w,i}$ to $A$.

After $A$ is done making queries, $A$ outputs $b'$ which $B$ also outputs. If the experiment for $A$ aborts for any reason, $B$ instead outputs 0. Observe that if $B$ received only ciphertexts and function keys for the first message or function of each of its challenge pairs, then $B$ exactly emulates $Hybrid_{1,w,4,k,4}^A$, and if $B$ received only ciphertexts and function keys for the second message or function of each of its challenge pairs, then $B$ emulates $Hybrid_{1,w,4,k+1,0}^A$. Additionally, $B$ does not need to know $FPFE_{msk_w}$ to carry out this experiment and makes only $Q$ function queries. Thus, by Equation 13, this means that $B$ breaks the $Q$-bounded, function-private, function-selective-IND-security of $FPFE$ as $B$ can distinguish between the two security games with non-negligible probability. \qed
Hybrid\textsuperscript{4}_{1,w,5}(\lambda) : This is identical to Hybrid\textsuperscript{4}_{1,w,4,Q}. Observe that we have now set each $H_{j,w}$ to $H'[\text{One-sFE.msk}_{j,w}, \text{One-sFE.Enc}_{j,w}, \text{PRF2}.K_{j,w}]$ which means that every function key will compute its output values for stream identity $w$ using $x^{(1)}_{w}$ instead of $x^{(0)}_{w}$.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter $1^\lambda$, 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. $\text{SKE}.sk \leftarrow \text{SKE.Setup}(1^\lambda)$.
     b. $\text{FE}.msk \leftarrow \text{FE.Setup}(1^\lambda)$.
     c. $\text{FE}.ek = \text{FE}.msk$.
   - **Public-Key Setting**:
     a. $\text{SKE}.sk \leftarrow \text{SKE.Setup}(1^\lambda)$.
     b. $(\text{FE}.mpk, \text{FE}.msk) \leftarrow \text{FE.Setup}(1^\lambda)$.
     c. $\text{FE}.ek = \text{FE}.mpk$.
     d. Send $\text{MPK} = \text{FE}.mpk$ to the adversary.

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

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

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. $\text{One-sFE.sk}_{f_{j,w}} \leftarrow \text{One-sFE.KeyGen}(\text{One-sFE.msk}_{j,w}, f_j)$.
      ii. $c_{j,w} \leftarrow \text{SKE.Enc}(\text{SKE}.sk, (\text{One-sFE.sk}_{f_{j,w}}, \text{FPFE.sk}_{H_{j,w}}))$.
      iii. Let $G_j = G[f_j, s_j, c_{j,w}]$ as defined in Figure 9 (page 50).
      iv. $\text{FE.sk}_{G_j} \leftarrow \text{FE.KeyGen}(\text{FE.msk}, G_j)$.
      v. Send $\text{SK}_{f_j} = \text{FE.sk}_{G_j}$ to the adversary.
(b) **Message Query:** For the \(i\)th message query made to stream identity \(id\), \(\mathcal{A}\) outputs a message pair \((x_{id,i}^{(0)}, x_{id,i}^{(1)})\) where \(x_{id,i}^{(0)}, x_{id,i}^{(1)} \in \{0, 1\}^\ell\).

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

ii. If \(id < w\), \(FPFE.ct_{id,i} \leftarrow FPFE.Enc(FPFE.msk_{id}, (i, t_{id,i}, x_{id,i}^{(1)}, 0^\ell, 0^\ell_{\text{One-sFE}}))\).

iii. If \(id = w\),
   - A. \(r_{Q,w,i} \leftarrow \text{PRF2.Eval(\text{PRF2}.K_{Q,w}, t_{w,i})}\).
   - B. \(v_{Q,w,i} \leftarrow \text{One-sFE.Enc(One-sFE.msk}_{Q,w}, \text{One-sFE.Enc.st}_{Q,w, i}, x_{w,i}^{(1)}, r_{Q,w,i})\).
   - C. \(FPFE.ct_{w,i} \leftarrow FPFE.Enc(FPFE.msk_{w}, (i, t_{w,i}, x_{w,i}^{(0)}, x_{w,i}^{(1)}, v_{Q,w,i}))\).

iv. If \(id > w\), \(FPFE.ct_{id,i} \leftarrow FPFE.Enc(FPFE.msk_{id}, (i, t_{id,i}, x_{id,i}^{(0)}, 0^\ell, 0^\ell_{\text{One-sFE}}))\).

v. If \(i = 1\), set \(CT_{id,1} = (FE.ct_{id}, FPFE.ct_{id,1})\). Else, set \(CT_{id,i} = FPFE.ct_{id,i}\).

vi. Send \(CT_{id,i}\) 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 \(w \in [\text{Bound}_A]\),

\[
\Pr[\text{Hybrid}_{1,w,4,Q,4}^A(1^\lambda) = 1] - \Pr[\text{Hybrid}_{1,w,5}^A(1^\lambda) = 1] = 0
\]

**Proof.** The hybrids are identical. \(\Box\)
**Hybrid4_{1,w,6}(1^λ):** 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 \( \text{FPFE} \cdot \text{ct}_{w,i} \) to an encryption of \((i, t_{w,i}, x_w^{(1)} \ell x, 0^\ell \text{One-sFE}, \alpha)\), and for every \( j \), we restore \( H_{j,w} \) to its original value.

This is the same as \( \text{Hybrid4}_{1,w,5} \) except that we change the following steps:

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

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

**Lemma 5.20.** If \( \text{FPFE} \) is \( Q \)-bounded, function-private, function-selective-IND-secure, then for all PPT adversaries \( A \) and all \( w \in [\text{Bound}_A] \),

\[
\left| \Pr[\text{Hybrid4}_{1,w,5}^A(1^\lambda) = 1] - \Pr[\text{Hybrid4}_{1,w,6}^A(1^\lambda) = 1] \right| \leq \text{negl}(\lambda)
\]

**Proof.** Suppose for sake of contradiction that there exists a PPT adversary \( A \) and \( w \in [\text{Bound}_A] \),

\[
\left| \Pr[\text{Hybrid4}_{1,w,5}^A(1^\lambda) = 1] - \Pr[\text{Hybrid4}_{1,w,6}^A(1^\lambda) = 1] \right| > \text{negl}(\lambda)
\]  

(14)

We build a PPT adversary \( B \) that breaks the \( Q \)-bounded, function-private, function-selective-IND-security of \( \text{FPFE} \). \( B \) first runs steps 1-3 of \( \text{Hybrid4}_{1,w,6} \) except that \( B \) does not compute \( \text{FPFE}.\text{msk}_w \).

For \( j \in [Q] \), \( B \) does the following: \( B \) computes \((s_j, \text{One-sFE}.\text{msk}_{j,w}, \text{One-sFE}.\text{Enc}.\text{st}_{j,w}, \text{PRF2}.K_{j,w})\) as in step 4 of \( \text{Hybrid4}_{1,w,6} \). \( B \) sets \( H_{j,w}^{(0)} = H(\text{One-sFE}.\text{msk}_{j,w}, \text{One-sFE}.\text{Enc}.\text{st}_{j,w}, \text{PRF2}.K_{j,w}) \) and \( H_{j,w}^{(1)} = H'(\text{One-sFE}.\text{msk}_{j,w}, \text{One-sFE}.\text{Enc}.\text{st}_{j,w}, \text{PRF2}.K_{j,w}) \).

- \( B \) sets its \( j^{th} \) challenge function pair to \((H_{j,w}^{(1)}, H_{j,w}^{(0)})\).
\( \mathcal{B} \) then sends all \( Q \) challenge function pairs to its FPFE challenger and receives \( \{ \text{FPFE} \text{.sk}_{H_{j,w}} \}_{j \in [Q]} \).

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

- For each message query \((x_{\text{id},i}^{(0)}, x_{\text{id},i}^{(1)})\) output by \( \mathcal{A} \):
  - If \( \text{id} \neq w \), \( \mathcal{B} \) computes \( \text{CT}_{\text{id},i} \) as in step 5a of \( \text{Hybrid}_{1,w,6}^A \), and sends \( \text{CT}_{\text{id},i} \) to \( \mathcal{A} \).
  - If \( \text{id} = w \), \( \mathcal{B} \) computes \((t_{w,i}, v_{Q,w,i})\) as in step 5b of \( \text{Hybrid}_{1,w,5}^A \). \( \mathcal{B} \) sends challenge message pair \(((i, t_{w,i}, x_{w,i}^{(0)}, x_{w,i}^{(1)}, v_{k,w,i}), (i, t_{w,i}, x_{w,i}^{(1)}, 0^\ell x, 0^{\text{FPFE.}\alpha}))\) to its FPFE challenger and receives \( \text{FPFE.ct}_{w,i} \). This is a valid message query since for all \( j \),
    \[
    H_{j,w}^{(1)}(i, t_{w,i}, x_{w,i}^{(0)}, x_{w,i}^{(1)}, v_{Q,w,i}) = H_{j,w}^{(0)}(i, t_{w,i}, x_{w,i}^{(1)}, 0^\ell x, 0^{\text{FPFE.}\alpha})
    \]
  - If \( i = 1 \), \( \mathcal{B} \) sets \( \text{CT}_{w,1} = (\text{FE.ct}_{w,1}, \text{FPFE.ct}_{w,1}) \). Else, \( \mathcal{B} \) sets \( \text{CT}_{w,i} = \text{FPFE.ct}_{w,i} \). \( \mathcal{B} \) sends \( \text{CT}_{w,i} \) 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 \( \text{Hybrid}_{1,w,5}^A \), 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 \( \text{Hybrid}_{1,w,6}^A \). Additionally, \( \mathcal{B} \) does not need to know \( \text{FPFE.msk}_w \) 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. 

\[ \square \]
**Hybrid\(^4\)\(_{1,w,7}(1^\lambda)\):** We revert back to using \(\text{PRF}.K_w\) to compute the randomness needed for One-sFE, PRF2, and FPFE.KeyGen on stream identity \(w\).

This is the same as **Hybrid\(^4\)\(_{1,w,6}\) except that we change the following steps:

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

5a. **Function Query:** For the \(j^{th}\) function query \(f_j \in F[\ell_F, \ell_S, \ell_X, \ell_Y]\) made by the adversary:
   
   (a) \(\text{One-sFE.sk}_{f_j,w} \leftarrow \text{One-sFE.KeyGen}(\text{One-sFE.msk}_{j,w}, f_j; r_{\text{KeyGen},j,w})\).
   (b) \(c_{j,w} \leftarrow \text{SKE.Enc}($\text{SKE.sk}, (\text{One-sFE.sk}_{f_j,w}, \text{FPFE.sk}_{H,j,w}))\).
   (c) Let \(G_j = G[f_j, s_j, c_{j,w}]\) as defined in Figure 9 (page 50).
   (d) \(\text{FE.sk}_{G_j} \leftarrow \text{FE.KeyGen}(\text{FE.msk}, G_j)\).
   (e) Send \(\text{SK}_{f_j} = \text{FE.sk}_{G_j}\) to the adversary.

**Lemma 5.21.** If \(\text{PRF}\) is a secure PRF, then for all PPT adversaries \(A\) and all \(w \in [\text{Bound}_A]\),

\[\Pr[\text{Hybrid}^4_{1,w,6}(1^\lambda) = 1] - \Pr[\text{Hybrid}^4_{1,w,7}(1^\lambda) = 1] \leq \text{negl}(\lambda)\]

**Proof.** This proof is essentially the same as the proof of Lemma 5.12. \(\square\)
**Hybrid**_{1,w,8}(1^{\lambda})$: We change the message encrypted in FE.\text{ct}_w back to its original value. This is the same as **Hybrid**_{1,w,7} except that we change the following steps:

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

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

\[
\Pr[\text{Hybrid}_{1,w,7}(1^{\lambda}) = 1] - \Pr[\text{Hybrid}_{1,w,8}(1^{\lambda}) = 1] \leq \text{negl}(\lambda)
\]

**Proof.** This proof is essentially the same as the proof of Lemma 5.11. 

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

This is the same as Hybrid$^A_{1,w,8}$ except that we change the following steps:

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


5a. **Function Query**: For the $j^{th}$ function query $f_j \in \mathcal{F}[\ell_F, \ell_S, \ell_X, \ell_Y]$ 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) $\text{FE}.sk_{G_j} \leftarrow \text{FE}.\text{KeyGen}(\text{FE}.\text{msk}, G_j)$.
   (e) Send $\text{SK}_{f_j} = \text{FE}.sk_{G_j}$ to the adversary.

**Lemma 5.23.** If SKE has pseudorandom ciphertexts, then for all PPT adversaries $A$ and all $w \in [\text{Bound}_A]$, 
\[
\left| \Pr[\text{Hybrid}^A_{1,w,8}(1^\lambda) = 1] - \Pr[\text{Hybrid}^A_{1,w,9}(1^\lambda) = 1] \right| \leq \text{negl}(\lambda)
\]

**Proof.** This proof is essentially the same as the proof of Lemma 5.10. \qed

**Lemma 5.24.** For all adversaries $A$ and all $w \in [\text{Bound}_A - 1]$, 
\[
\left| \Pr[\text{Hybrid}^A_{1,w,8}(1^\lambda) = 1] - \Pr[\text{Hybrid}^A_{1,w+1,0}(1^\lambda) = 1] \right| = 0
\]

**Proof.** The hybrids are identical. \qed
**Hybrid**$^4_2(1^\lambda)$: This is the real world experiment with $b = 1$. This is identical to Hybrid$^4_{1,Bound_A,9}$.

1. **Parameters**: The adversary $A$ receives security parameter $1^\lambda$, and outputs a function size $1^{\ell_F}$, a state size $1^{\ell_S}$, an input size $1^{\ell_X}$, and an output size $1^{\ell_Y}$.

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

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

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 $f_j \in F[\ell_F, \ell_S, \ell_X, \ell_Y]$ made by the adversary:  
   i. $s_j \leftarrow \{0, 1\}^\lambda$.  
   ii. $c_j \leftarrow \{0, 1\}^{\ell_{SKE.msk}}$.  
   iii. Let $G_j = G[f_j, s_j, c_j]$ as defined in Figure 9 (page 50).  
   iv. $FE.sk_{G_j} \leftarrow FE.KeyGen(FE.msk, G_j)$.  
   v. Send $SK_{f_j} = FE.sk_{G_j}$ to the adversary.

   (b) **Message Query**: For the $i^{th}$ message query made to stream identity $id$, $A$ outputs a message pair $(x_{id,i}^{(0)}, x_{id,i}^{(1)})$ where $x_{id,i}^{(0)}, x_{id,i}^{(1)} \in \{0, 1\}^{\ell_X}$.  
   i. $t_{id,i} \leftarrow \{0, 1\}^\lambda$.  
   ii. $FPFE.ct_{id,i} \leftarrow FPFE.Enc(FPFE.msk_{id}, (i, t_{id,i}, x_{id,i}^{(1)}, 0^{\ell_X}, 0^{\ell_{OneBits.FE.ct}}))$.  
   iii. If $i = 1$, set $CT_{id,1} = (FE.ct_{id}, FPFE.ct_{id,1})$. Else, set $CT_{id,i} = FPFE.ct_{id,i}$.  
   iv. Send $CT_{id,i}$ to the adversary.

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

**Lemma 5.25.** For all adversaries $A$,  
$$\left| \Pr[Hybrid^4_{1,Bound_A,9}(1^\lambda) = 1] - \Pr[Hybrid^4_2(1^\lambda) = 1] \right| = 0$$

*Proof.* The hybrids are identical.  

80
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 PRF\(^2\) 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


A Preliminaries Continued

A.1 Standard Notions

Definition A.1 (Pseudorandom Function (PRF)). A pseudorandom function family (PRF) with key space \( K = \{ K_{\lambda,n,m} \}_{\lambda,n,m \in \mathbb{N}} \) is a tuple of PPT algorithms \( \text{PRF} = (\text{PRF}.\text{Setup}, \text{PRF}.\text{Eval}) \) where

- \( \text{PRF}.\text{Setup}(1^\lambda, 1^n, 1^m) \) is a randomized algorithm that takes as input the security parameter \( \lambda \), an input length \( n \), and an output length \( m \), and outputs a key \( K \in K_{\lambda,n,m} \).
- \( \text{PRF}.\text{Eval}(K, x) \) is a deterministic algorithm that takes as input a key \( K \in K_{\lambda,n,m} \) 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 \( \mu \) such that for all \( \lambda \in \mathbb{N} \) and all PPT adversaries \( A \),

\[
\left| \Pr[A^{\text{PRF}_{\lambda,n,m}}(1^\lambda, 0) = 1] - \Pr[A^{\text{PRF}_{\lambda,n,m}}(1^\lambda, 1) = 1] \right| \leq \mu(\lambda)
\]

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

\[
\text{Expt}^\text{PRF}_{A}(1^\lambda, b)
\]

1. Parameters: \( A \) takes as input \( 1^\lambda \) and outputs an input size \( 1^n \) and an output size \( 1^m \).

2. Setup:
   (a) If \( b = 0 \), sample \( K \leftarrow \text{PRF}.\text{Setup}(1^\lambda, 1^n, 1^m) \).
   (b) If \( b = 1 \), sample \( R \leftarrow R_{n,m} \) where \( R_{n,m} \) 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) \( A \) outputs a value \( x \in \{ 0, 1 \}^n \).
   (b) If \( b = 0 \), send \( y = \text{PRF}.\text{Eval}(K, x) \) to \( A \).
   (c) If \( b = 1 \), send \( y = R(x) \) to \( A \).

4. Experiment Outcome: \( A \) 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 \( K = \{ K_\lambda \}_\lambda \) and ciphertext size \( m(\cdot) \) is a tuple of PPT algorithms \( \text{SKE} = (\text{SKE}.\text{Setup}, \text{SKE}.\text{Enc}, \text{SKE}.\text{Dec}) \) where

- \( \text{SKE}.\text{Setup}(1^\lambda) \) is a randomized algorithm that takes as input the security parameter \( \lambda \) and outputs a secret key \( sk \in K_\lambda \).
• **SKE.Enc**($\text{sk}, x$) is a randomized algorithm that takes as input a secret key $\text{sk} \in \mathcal{K}_{\lambda,n}$ and a message $x \in \{0, 1\}^*$ and outputs an encryption $\text{ct} \in \{0, 1\}^{m(\lambda,|x|)}$ of $x$.

• **SKE.Dec**($\text{sk}, \text{ct}$) is a deterministic algorithm that takes as input a secret key $\text{sk} \in \mathcal{K}_{\lambda}$ and a ciphertext $\text{ct} \in \{0, 1\}^{m(\lambda,n)}$ for some $n$ and outputs a value $y \in \{0, 1\}^n$.

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

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

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

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

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

\[
\text{Expt}_{A}^{\text{SKE}(1^\lambda, b)}
\]

1. **Parameters**: $A$ takes as input $1^\lambda$.

2. **Setup**: $\text{sk} \leftarrow \text{SKE.Setup}(1^\lambda)$.

3. **Challenge Message Queries**: The following can be repeated any polynomial number of times:
   
   (a) $A$ outputs a challenge message pair $(x_0, x_1)$ where $|x_0| = |x_1|$.
   
   (b) $\text{ct}_b \leftarrow \text{SKE.Enc}($sk, $x_b$)
   
   (c) Sent $\text{ct}_b$ to $A$.

4. **Experiment Outcome**: $A$ 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.

\textbf{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 $\mu$ such that for all $\lambda \in \mathbb{N}$ and every PPT adversary $A$,

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

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

\[
\text{Expt}_{A}^{\text{SKE-Pseudorandom-CT}(1^\lambda, b)}
\]

1. **Parameters**: $A$ takes as input $1^\lambda$.

2. **Setup**: $\text{sk} \leftarrow \text{SKE.Setup}(1^\lambda)$

3. **Challenge Message Queries**: The following can be repeated any polynomial number of times:
   
   (a) $A$ outputs a challenge message $x$ where $x \in \{0, 1\}^*$.
\( (b) \) If \( b = 0 \), \( ct \leftarrow \text{SKE.Enc}(sk, x) \).
\( (c) \) If \( b = 1 \), \( ct \leftarrow \{0, 1\}^{\mu(\lambda|x)} \)
\( (d) \) Send \( ct \) to \( A \).

4. **Experiment Outcome:** \( A \) 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 \( \text{P/Poly} \) is a tuple of PPT algorithms \( \text{FE} = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec}) \) defined as follows:\(^{21}\)

- **Setup**\( (1^\lambda, 1^{\ell_F}, 1^{\ell_X}, 1^{\ell_Y}) \): takes as input the security parameter \( \lambda \), a function size \( \ell_F \), an input size \( \ell_X \), and an output size \( \ell_Y \), and outputs the master secret key \( \text{msk} \).
- **Enc**\( (\text{msk}, x) \): takes as input the master secret key \( \text{msk} \) and a message \( x \in \{0, 1\}^{\ell_X} \), and outputs an encryption \( ct \) of \( x \).
- **KeyGen**\( (\text{msk}, f) \): takes as input the master secret key \( \text{msk} \) and a function \( f \in \mathcal{F}[\ell_F, \ell_X, \ell_Y] \), and outputs a function key \( \text{sk}_f \).
- **Dec**\( (\text{sk}_f, ct) \): takes as input a function key \( \text{sk}_f \) and a ciphertext \( ct \), and outputs a value \( y \in \{0, 1\}^{\ell_Y} \).

\( \text{FE} \) satisfies **correctness** if for all polynomials \( p \), there exists a negligible function \( \mu \) such that for all \( 1^\lambda \in \mathbb{N} \), all \( \ell_F, \ell_X, \ell_Y \leq p(1^\lambda) \), all \( x \in \{0, 1\}^{\ell_X} \), and all \( f \in \mathcal{F}[\ell_F, \ell_X, \ell_Y] \),

\[
\Pr \left[ \text{Dec}(\text{sk}_f, \text{ct}_x) = f(x) : \text{ct}_x \leftarrow \text{Enc}(\text{msk}, x) \right] \geq 1 - \mu(\lambda).
\]

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

\[
\left| \Pr_{\text{SKExpt}^{\text{FE-Q-Ad-IND}}_A(1^\lambda, 0) = 1} - \Pr_{\text{SKExpt}^{\text{FE-Q-Ad-IND}}_A(1^\lambda, 1) = 1} \right| \leq \mu(1^\lambda)
\]

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

\( \text{SKExpt}^{\text{FE-Q-Ad-IND}}_A(1^\lambda, b) \)

1. **Parameters:** \( A \) takes as input \( 1^\lambda \), and outputs a function size \( 1^{\ell_F} \), an input size \( 1^{\ell_X} \), and an output size \( 1^{\ell_Y} \).
2. **Setup:** \( \text{msk} \leftarrow \text{FE.Setup}(1^\lambda, 1^{\ell_F}, 1^{\ell_X}, 1^{\ell_Y}) \).
3. For a polynomial number of rounds, the adversary can do either one of the following in each round:
   - **Function Query:** The adversary can make at most \( Q = Q(\lambda) \) such queries:

\(^{21}\)We also allow \( \text{Enc}, \text{KeyGen}, \) and \( \text{Dec} \) to additionally receive parameters \( 1^\lambda, 1^{\ell_F}, 1^{\ell_X}, 1^{\ell_Y} \) as input, but omit them from our notation for convenience.
A outputs a function query \( f \in \mathcal{F}[\ell_F, \ell_X, \ell_Y] \)

\( \text{sk}_f \leftarrow \text{FE.KeyGen}(\text{msk}, f) \)

Send \( \text{sk}_f \) to \( A \)

(b) **Message Query:**

i. \( A \) outputs a message pair \((x_0, x_1)\) where \( x_0, x_1 \in \{0, 1\}^{\ell_X} \).

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

iii. Send \( \text{ct} \) to \( A \).

4. **Experiment Outcome:** \( A \) 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 \( \text{FE} \) for \( \text{P/Poly} \) is Q-bounded, function-private, function-selective-IND-secure if there exists a negligible function \( \mu \) such that for all \( \lambda \in \mathbb{N} \) and every PPT adversary \( A \),

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

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

\[
\text{SKExpt}_A^{\text{FE}-Q-\text{FuncPriv-FuncSel-IND}}(1^\lambda, b)
\]

1. **Parameters:** \( A \) takes as input \( 1^\lambda \), and outputs a function size \( 1^{\ell_F} \), an input size \( 1^{\ell_X} \), and an output size \( 1^{\ell_Y} \).

2. **Setup:** \( \text{msk} \leftarrow \text{FE.Setup}(1^\lambda, 1^{\ell_F}, 1^{\ell_X}, 1^{\ell_Y}) \).

3. **Function Queries:** The following can be repeated at most \( Q = Q(\lambda) \) times:

   (a) \( A \) outputs a function query pair \((f_0, f_1)\) where \( f_0, f_1 \in \mathcal{F}[\ell_F, \ell_X, \ell_Y] \)

   (b) \( \text{sk}_f \leftarrow \text{FE.KeyGen}(\text{msk}, f_b) \)

   (c) Send \( \text{sk}_f \) to \( A \)
4. **Message Queries**: The following can be repeated any polynomial number of times:

   (a) $A$ outputs a message pair $(x_0, x_1)$ where $x_0, x_1 \in \{0, 1\}^{\ell_X}$.
   (b) $ct \leftarrow FE.Enc(msk, x_0)$.
   (c) Send $ct$ to $A$.

5. **Experiment Outcome**: $A$ 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 P/Poly is a tuple of PPT algorithms $sFE = (\text{Setup}, \text{EncSetup}, \text{Enc}, \text{KeyGen}, \text{Dec})$ defined as follows:\textsuperscript{22}

1. $\text{Setup}(1^\lambda, 1^{\ell_F}, 1^{\ell_S}, 1^{\ell_X}, 1^{\ell_Y})$: takes as input the security parameter $\lambda$, a function size $\ell_F$, a state size $\ell_S$, an input size $\ell_X$, and an output size $\ell_Y$, and outputs the master secret key $msk$.

2. $\text{EncSetup}(msk)$: takes as input the master secret key $msk$ and outputs an encryption state $Enc.st$.

3. $\text{Enc}(msk, Enc.st, i, x_i)$: takes as input the master secret key $msk$, an encryption state $Enc.st$, an index $i$, and a message $x_i \in \{0, 1\}^{\ell_X}$ and outputs an encryption $ct_i$ of $x_i$.

4. $\text{KeyGen}(msk, f)$: takes as input the master secret key $msk$ and a function $f \in \mathcal{F}[\ell_F, \ell_S, \ell_X, \ell_Y]$ and outputs a function key $sk_f$.

5. $\text{Dec}(sk_f, Dec.st_i, i, ct_i)$: where for each function key $sk_f$, $\text{Dec}(sk_f, \cdot, \cdot, \cdot)$ is a streaming function that takes as input a state $Dec.st_i$, an index $i$, and an encryption $ct_i$ and outputs a new state $Dec.st_{i+1}$ and an output $y_i \in \{0, 1\}^{\ell_Y}$.

$sFE$ must be **streaming efficient**, meaning that the size and runtime of all algorithms of $sFE$ on security parameter $\lambda$, function size $\ell_F$, state size $\ell_S$, input size $\ell_X$, and output size $\ell_Y$ are $\text{poly}(\lambda, \ell_F, \ell_S, \ell_X, \ell_Y)$.

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

$$\Pr \left[ \text{Dec}(sk_f, ct_x) = f(x) : \begin{array}{c}
msk \leftarrow \text{Setup}(1^\lambda, 1^{\ell_F}, 1^{\ell_S}, 1^{\ell_X}, 1^{\ell_Y}), \\
c_{t} \leftarrow \text{Enc}(msk, x), \\
sk_f \leftarrow \text{KeyGen}(msk, f)
\end{array} \right] \geq 1 - \mu(\lambda)$$

where we define\textsuperscript{23}

\textsuperscript{22}We also allow $\text{Enc}, \text{EncSetup}, \text{KeyGen}$, and $\text{Dec}$ to additionally receive parameters $1^\lambda, 1^{\ell_F}, 1^{\ell_S}, 1^{\ell_X}, 1^{\ell_Y}$ as input, but omit them from our notation for convenience.

\textsuperscript{23}As with all streaming functions, we assume that $\text{Dec.st}_1 = \perp$ if not otherwise specified.
- $\text{Enc}(\text{msk}, x)$ outputs $\text{ct}_x = (\text{ct}_i)_{i \in [n]}$ produced by sampling $\text{Enc}.\text{st} \leftarrow \text{EncSetup}(\text{msk})$ and then computing $\text{ct}_i \leftarrow \text{Enc}(\text{msk}, \text{Enc}.\text{st}, i, x_i)$ for $i \in [n]$.

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

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

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

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

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

1. **Parameters:** $\mathcal{A}$ takes as input $1^\lambda$, and outputs a function size $1^{\ell_f}$, a state size $1^{\ell_s}$, an input size $1^{\ell_x}$, and an output size $1^{\ell_y}$.

2. **Setup:** Compute $\text{msk} \leftarrow \text{sFE}.\text{Setup}(1^\lambda, 1^{\ell_f}, 1^{\ell_s}, 1^{\ell_x}, 1^{\ell_y})$.

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 $f \in \mathcal{F}(\ell_f, \ell_s, \ell_x, \ell_y)$.
      
      ii. $\text{sk}_f \leftarrow \text{sFE}.\text{KeyGen}(\text{msk}, f)$.
      
      iii. Send $\text{sk}_f$ to $\mathcal{A}$.

   (b) **Message Query:**
      
      i. $\mathcal{A}$ outputs a stream identity $\text{id}$.
         
         a. If this is the first message query with stream identity $\text{id}$, sample $\text{Enc}.\text{st}_\text{id} \leftarrow \text{sFE}.\text{EncSetup}(\text{msk})$ and initialize $\text{index}_{\text{id}} = 1$. Else, increment $\text{index}_{\text{id}}$ by 1.
      
      b. Set $i = \text{index}_{\text{id}}$.
      
      ii. $\mathcal{A}$ outputs a message pair $(x^{(0)}_{\text{id}, i}, x^{(1)}_{\text{id}, i})$ for stream identity $\text{id}$ where $x^{(0)}_{\text{id}, i}, x^{(1)}_{\text{id}, i} \in \{0, 1\}^{\ell_y}$.
      
      iii. $\text{ct}_{\text{id}, i} \leftarrow \text{sFE}.\text{Enc}(\text{msk}, \text{Enc}.\text{st}_{\text{id}}, i, x^{(b)}_{\text{id}, i})$.
      
      iv. Send $\text{ct}_{\text{id}, i}$ 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 $f(x^{(0)}_{\text{id}}) \neq f(x^{(1)}_{\text{id}})$ for some function query $f$ submitted by the adversary where $\{(x^{(0)}_{\text{id}, i}, x^{(1)}_{\text{id}, i})\}_{i \in [t]}$ are the message queries submitted so far under some stream identity $\text{id}$, $x^{(0)}_{\text{id}} = x^{(0)}_{\text{id}, 1} \ldots x^{(0)}_{\text{id}, t}$, and $x^{(1)}_{\text{id}} = x^{(1)}_{\text{id}, 1} \ldots x^{(1)}_{\text{id}, t}$).

**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 $P/Poly$ is single-key, single-ciphertext, function-selective-SIM-secure if there exists a PPT simulator $Sim$ and a negligible function $\mu$ such that for all $\lambda \in \mathbb{N}$ and all PPT adversaries $A$,

$$\Pr[RealExpt_{A,Sim}^{One-sFE-SIM}(1^\lambda) = 1] - \Pr[IdealExpt_{A,Sim}^{One-sFE-SIM}(1^\lambda) = 1] \leq \mu(\lambda)$$

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

\[
\text{RealExpt}_{A}^{One-sFE-SIM}(1^\lambda)
\]

1. **Parameters**: $A$ takes as input $1^\lambda$, and outputs a function size $1^{\ell_f}$, a state size $1^{\ell_s}$, an input size $1^{\ell_x}$, and an output size $1^{\ell_y}$.

2. **Setup**: $msk \leftarrow sFE.\text{Setup}(1^\lambda, 1^{\ell_f}, 1^{\ell_s}, 1^{\ell_x}, 1^{\ell_y})$

3. **Function Query**:
   - (a) $A$ outputs a streaming function query $f \in \mathcal{F}[\ell_f, \ell_s, \ell_x, \ell_y]$.
   - (b) $sk_f \leftarrow sFE.\text{KeyGen}(msk, f)$
   - (c) Send $sk_f$ to $A$.

4. **Challenge Message Queries**: The following can be repeated any polynomial number of times:
   - (a) If this is the first challenge message query, sample $Enc.st \leftarrow sFE.\text{EncSetup}(msk)$ and initialize the index $i = 1$. Else, increment the index $i$ by 1.
   - (b) $A$ outputs a challenge message $x_i \in \{0, 1\}^{\ell_x}$.
   - (c) $ct_i \leftarrow sFE.\text{Enc}(msk, Enc.st, i, x_i)$.
   - (d) Send $ct_i$ to $A$.

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

\[
\text{IdealExpt}_{A,Sim}^{One-sFE-SIM}(1^\lambda)
\]

1. **Parameters**: $A$ takes as input $1^\lambda$, and outputs a function size $1^{\ell_f}$, a state size $1^{\ell_s}$, an input size $1^{\ell_x}$, and an output size $1^{\ell_y}$. $Sim$ receives $(1^\lambda, 1^{\ell_f}, 1^{\ell_s}, 1^{\ell_x}, 1^{\ell_y})$. 

92
2. **Function Query:**

   (a) \( A \) outputs a streaming function query \( f \in \mathcal{F}[\ell_F, \ell_S, \ell_X, \ell_Y] \).
   (b) \( \text{Sim} \) receives \( f \) and outputs a function key \( \text{sk}_f \).
   (c) Send \( \text{sk}_f \) to \( A \).

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 \( \text{st}_1 = \perp \).
   Else, increment \( i \) by 1.
   (b) \( A \) outputs a message \( x_i \in \{0, 1\}^{\ell_X} \).
   (c) \( (y_i, \text{st}_{i+1}) = f(x_i, \text{st}_i) \).
   (d) \( \text{Sim} \) receives \( y_i \) and outputs a ciphertext \( \text{ct}_i \).
   (e) Send \( \text{ct}_i \) to \( A \).

4. **Experiment Outcome:** \( A \) 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.