Multi-Hop Functional Encryption

Pratish Datta* Jiaxin Guan† Alexis Korb‡ Amit Sahai§

Abstract

In this paper, we initiate the study of *multi-hop functional encryption* (MHFE), a new primitive for computing over encrypted data in recursive and multi-stage workflows while enabling controlled release of intermediate outputs. MHFE supports dynamic directed-acyclic computation graphs in which each node is labeled by a function and is equipped with a restricted secret key that reveals only that node's output, yet the computation can continue downstream on a forwardable encrypted state. In doing so, MHFE simultaneously overcomes (i) multi-hop fully homomorphic encryption's need for a globally trusted, always-online decryption key to reveal intermediate values, (ii) standard FE's lack of a native mechanism for forwarding reusable encrypted state across stages and handling evolving datasets without a fresh recomputation, and (iii) Multi Party Computation's deployment friction stemming from interaction and liveness/coordination requirements (at the cost of a central key-issuing authority).

We formalize MHFE security via the simulation paradigm extending standard FE collusion resistance to the multi-hop setting; this simulation-based notion captures the intended information flow in recursive computations most accurately by characterizing an adversary's entire view as simulatable given only the authorized intermediate outputs. Our security games impose a natural *topological-order* on queries: a hop key for a node may be requested only after the ciphertexts/keys on all incoming edges to that node have been fixed. We then give the *first* generic *simulation-secure* construction of public-key MHFE for all polynomial-size circuits from any IND-secure standard FE for circuits.

This article is the full version of the article under the same name appearing at TCC 2026, submitted by the authors to the IACR and to Springer-Verlag on September 15, 2026. The version published by Springer-Verlag is available at `https://doi.org/10.1007/978-3-032-43630-6_18`.

*NTT Research. Email: `pratish.datta@ntt-research.com`. ORCID: `0000-0002-3938-7594`.

†Zellic. Email: `jiaxin@guan.io`. ORCID: `0000-0003-1823-8845`.

‡Pepperdine University. Email: `alexis.korb@pepperdine.edu`. ORCID: `0000-0001-6888-5296`. This research was conducted while this author was a Ph.D. student at UCLA.

§UCLA. Email: `sahai@cs.ucla.edu`. ORCID: `0000-0003-2216-9600`.

Contents

1 Introduction **3**
- 1.1 Multi-Hop Functional Encryption . . . 4
- 1.2 Our Results . . . 5
- 1.3 Why Existing Primitives Fall Short . . . 7

2 Technical Overview **8**
- 2.1 Natural Construction Attempts . . . 11
- 2.2 Our Approach . . . 13

3 Preliminaries **18**
- 3.1 Standard Notions . . . 18
- 3.2 Public-Key Functional Encryption . . . 21
- 3.3 Function-Private Secret-Key Functional Encryption . . . 22

4 Definitions **23**
- 4.1 Multi-Hop Functional Encryption (MHFE) . . . 23
- 4.2 Multi-Hop Garbled Circuits (MHGC) . . . 31

5 Constructing a Multi-Hop Garbled Circuit **34**
- 5.1 Parameters . . . 34
- 5.2 Construction . . . 36
- 5.3 Correctness . . . 37
- 5.4 Security . . . 37

6 Bootstrapping to a Multi-Hop Functional Encryption Scheme **50**
- 6.1 Parameters . . . 51
- 6.2 Construction . . . 53
- 6.3 Correctness . . . 55
- 6.4 Security . . . 57

7 Acknowledgements **76**

8 References **77**

1 Introduction

In the modern world, there is an increasing demand for the ability to perform secure computation on encrypted data. Computation is often performed by parties who, for privacy or security reasons, may not be allowed full access to view the data, but may be given limited authority to compute on and learn certain specific functions of the data. At the same time, data workflows have become increasingly more complex, challenging the ability of existing cryptographic primitives to keep up with modern needs. Computation is often distributed across multiple parties that have limited availability and interactions patterns. Moreover, the path the data takes between these parties may be determined on the fly, rather than fixed in advance.

Consider the following scenario. A federal government needs to periodically audit financial records that were relevant to government operations. However, different government workers may need to process different sets of financial records. For example, a government office may desire aggregate statistics for census purposes, a state oversight office may wish to look for abnormalities in spending, or a contractor may need to compute its reimbursement total. Additionally, some records contain especially sensitive items (e.g. personally identifiable information about government employees or detailed transaction information with vendors). There may be no single party that is authorized to review and assess all the data and determine what information is needed by each party. One cryptographic solution is for the government to use Functional Encryption (FE) [BSW11, O'N10]. This allows the government to encrypt their data just once, and then give each party a restricted functional key which enables each party to compute on the encrypted data and learn only a specific allowed function of the data.

While standard FE enables this type of secure interaction, it requires that each party's private outputs be computed *from scratch*, starting with the original encrypted data. However, many natural data workflows are incremental or hierarchical. Consider our previous example of a government that encrypts its financial records. Now, suppose a state auditor wishes to audit a random subset of financial records that were used by government workers to complete their duties. To save work for the state, we would expect county workers to keep track of the financial records that were relevant to their work. However, county workers may not be authorized to view all of the information needed in those records and thus we want the data to remain encrypted while it is being computed upon by county workers. Thus we may wish to use (multi-hop) homomorphic encryption [GHV10] which allows intermediate users in the network to compute on encrypted data without decrypting it, and this encrypted data could then be passed along to the state auditor.

The downside of using homomorphic encryption here is that any intermediate users (e.g. the county workers) would not be able to learn any output information of their own during the computation process. All data remains encrypted while they are computing on it. However, as noted above, county workers also have legitimate reasons to learn specific functions of the encrypted data. Furthermore, the computation they perform for their own purposes may heavily overlap with the computation they need to do for the state authorities (such as selecting relevant financial records). Thus, to prevent them from having to do the work twice, we want a system that can combine the functionality of two different cryptographic objects:

(i) Computation of specific functions of encrypted data, as in functional encryption.

(ii) Continued computation on encrypted states, as in multi-hop homomorphic encryption.

Figure 1 depicts an abstraction of an example workflow. We use a tree diagram below, but our abstraction extends to general directed acyclic graph. Each node consumes encrypted values on its incoming edges, uses a restricted hop key to release only the output authorized for that node,

and simultaneously produces encrypted values on its internal outgoing edges. These internal-edge ciphertexts are not encryptions of the clear output; they are forwardable encrypted states, well formed as inputs to future hop keys and bound to the computation that produced them.

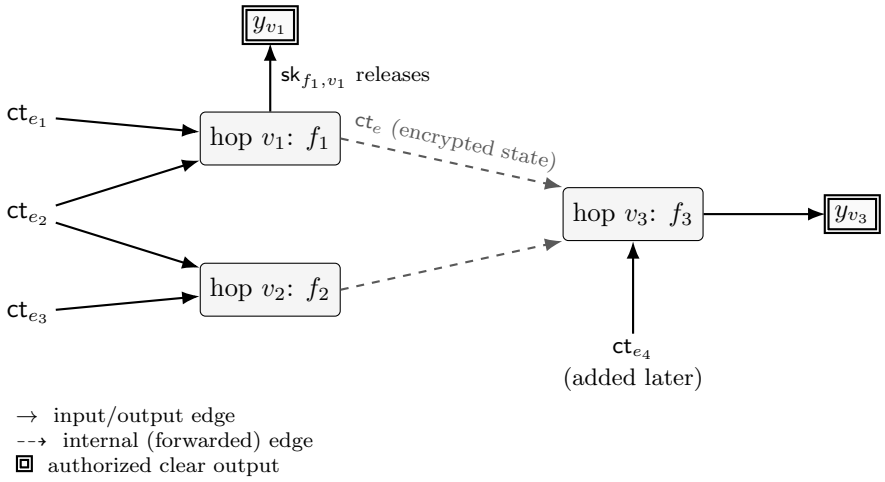

Figure 1: An abstract multi-hop encrypted workflow. Each hop is equipped with a restricted hop key sk_{f_i,v_i}, releases an authorized clear output y_{v_i}, and forwards encrypted state on internal edges to downstream hops. The future topology need not be fixed when the initial inputs are encrypted. New inputs (e.g., ct_{e_4}) and new hops can be incorporated as the workflow evolves.

Note that this abstraction captures many natural data workflows, in which there may be complex webs of dependency and computation, and where each participating party may have different levels of security clearance over the data. We emphasize three properties driving this work:

(i) *eliminating wasteful computation by reusing intermediate encrypted states,*

(ii) *supporting evolving datasets in which new inputs arrive and new analyses are launched after earlier inputs were encrypted, so the workflow extends rather than restarts,*

(iii) *role-specific authorization for sensitive data shared across distinct stakeholders,*

No existing primitive captures this combination—we explain why in Section 1.3.

1.1 Multi-Hop Functional Encryption

We introduce multi-hop functional encryption (MHFE), a public-key FE primitive for encrypted computation over a general network with selective intermediate output release. As in standard FE, computation proceeds via the issuance of function-specific secret keys, but each hop now corresponds to a node in a directed acyclic graph (DAG) whose topology may be determined dynamically as the workflow progresses. The authority issues a restricted hop key sk_f for each node, authorizing the local computation and output release. An encryptor can use the master public key to encrypt new ciphertexts, and it can further augment the ciphertext collection as new data arrives.[1] Given

[1]More specifically, the master public key can be used to produce an encryption key which is then used to encrypt messages belonging to the same dataset. The purpose of the encryption key is to prevent mix and match attacks: only ciphertexts encrypted under the same encryption key can be computed on together. This differentiates our scheme from multi-input FE which allows for an arbitrary set of ciphertexts to be combined together (perhaps from different encryptors), and allows us to avoid the strong impossibility results regarding simulation secure multi-input FE (which would be implied by our scheme if we allowed arbitrary mixing and matching of ciphertexts.)

sk_f, hop-decryption on the node's incoming-edge ciphertexts reveals only the authorized output $f(\cdot)$ on the corresponding plaintexts and, crucially, produces a processed encrypted state that is forwarded to the nodes along outgoing edges. Importantly, MHFE imposes no fixed topology. Hops can be composed and reused in arbitrary modular pipelines, with correctness guaranteeing that hop outputs remain decryptable and reusable in subsequent hops under appropriately issued keys. For security, we require that an adversary that obtains many hop keys, observes initial ciphertexts, runs hop decryptions, and sees all forwarded encrypted artifacts should learn nothing beyond the authorized clear outputs of the queried hop keys. We formalize this both in a simulation-based and in an indistinguishability-based paradigm, presented below:

Simulation-Based Security. Consider any adversary that is given ciphertexts encrypting some underlying dataset and a collection of hop keys for functions of its choice, possibly adaptively as the computation progresses, potentially spanning many hops and many computation DAGs. As the adversary applies these keys, it obtains the authorized intermediate clear functional outputs it is entitled to learn, and it may also see the next-hop ciphertexts/encrypted states produced to enable continued downstream computation. The security requirement asserts that the adversary's entire view can be simulated without access to the plaintexts, given only these authorized "clear" functional outputs; in other words, ciphertexts, keys, and all hop-by-hop artifacts should not leak anything beyond the prescribed clear hop outputs. Crucially, we impose a natural topological-order restriction on adaptivity: a hop key for a node may be requested and used only after the ciphertexts and keys on all incoming edges to that node have already been fixed in the adversary's view. We believe that this restriction is natural and reflects many real recursive workflows, while removing this restriction is also a central open problem left by our work. This simulation-based guarantee directly addresses the kinds of "learn nothing beyond what was authorized" concerns that arise in recursive encrypted workflows, and it is stronger and more descriptive than indistinguishability-based best-possible notions.

Indistinguishability-Based Security. We also consider a natural indistinguishability-based notion, mirroring the standard IND definition for FE [BSW11, O'N10]. The adversary receives encryptions of one of two candidate initial datasets and, subject to the same topological-order restriction, adaptively queries hop keys, learning each authorized hop output and the resulting next-hop ciphertexts. The adversary's challenge pair and adaptive hop-key usage are admissible if, for every hop key it obtains and every hop at which it is used, the revealed clear hop output is identical in the two challenge worlds; the IND requirement asserts that no efficient adversary making only admissible queries distinguishes the two worlds with non-negligible advantage. Just as in standard FE, our one-ciphertext, (many-topology) simulation security implies many-ciphertexts, (many-topology) indistinguishability security in the public-key setting via a standard bundle-by-bundle hybrid argument, simulating each challenge ciphertext bundle independently from only the authorized clear hop outputs (which agree on the two worlds by admissibility).

1.2 Our Results

This work initiates the study of MHFE as a primitive for recursive encrypted workflows with controlled intermediate disclosure, and gives the first generic public-key construction. Our contributions are the following.

I. Defining and Formalizing MHFE. A principal contribution of this paper is to formalize the notion of MHFE for polynomial-size circuits over dynamically constructed DAGs that natively support evolving datasets. New inputs may be encrypted at later times and new topologies grown

in a downstream fashion over the same initial encrypted data. The syntax separates encryption of input edges, hop-key generation, and hop decryption, and makes forwardable encrypted state a first-class output of decryption. Correctness states that evaluating hop keys along a valid DAG yields the same clear output-edge values as the underlying plaintext workflow while preserving encrypted internal-edge state for downstream hops.

II. Introducing Multi-Hop Garbled Circuits (MHGC). As the central technical stepping stone to MHFE, and as a primitive of independent interest, we introduce *multi-hop garbled circuits* (MHGC) [2]. MHGC can be viewed as a one-ciphertext, one-topology, secret-key analogue of MHFE, where "one ciphertext" means a single garbled encoding of a database/state carried across hops. At a high level, an MHGC scheme is described using the standard garbling syntax [BHR12] with separate procedures for circuit garbling and input encoding. A garbling algorithm produces, for a hop function, a garbled circuit together with encoding and decoding information; an input-encoding algorithm uses the encoding information to map a (possibly evolving) plaintext state to corresponding input labels; and an evaluation algorithm maps input labels to output labels. The defining multi-hop feature is that evaluation outputs not only the hop's clear outputs but also a forwardable encoded state, i.e., labels well-formed as inputs to subsequently garbled hop functions, enabling chained recursive evaluation on an evolving garbled state without ever materializing the full plaintext. Intermediate clear outputs are delivered via hop-specific decoding information, and revealing them does not disrupt continued evaluation on the forwardable state.

MHGC admits a natural simulation-based security definition along the same lines as MHFE. The adversary obtains garbled inputs and garbled circuits in topological order, and a simulator reproduces the entire view from only the authorized clear outputs.

III. Instantiating MHFE. Another central contribution of this work is the *first* instantiation of simulation-secure MHFE for general circuits in the public-key setting. We give a generic construction in which each node function can be an arbitrary polynomial-size circuit, assuming a standard public-key FE scheme for general circuits.

Our construction proceeds via two conceptual steps that may be of independent interest. First, we provide a simple construction of multi-hop garbling from standard garbled circuits.

Theorem 1.1 (Informal)**.** *Assuming one-way functions, there exists a simulation-secure* MHGC *scheme for* P/poly *in the plain model.*

A key conceptual choice in our framework is that garbled circuits arise as part of the keys encoding node functionality, rather than being embedded into ciphertexts as in prior uses of garbling for public-key primitives such as one-ciphertext, one-key simulation-secure public-key FE [SS10], and the DDH-based IBE construction of Döttling and Garg [DG17]. Placing garbled circuits inside hop keys is natural in the FE/MHFE setting: keys are the vehicle for functionality and authorization, and this cleanly decouples "encrypt once" data outsourcing from the (possibly evolving) choice of downstream computation DAGs.

[2] Reactive garbling [NR16] also targets stateful reactive computation over garbled values, but its known instantiations use random oracles and its port-wiring interface does not match the node-and-edge workflow needed here. For MHFE, MHGC gives a more direct abstraction: each node declares which outputs are clear and which remain internal encodings, matching the per-node leakage policy, while input encodings can be generated independently of any node circuit, enabling FE-derived encodings at decryption time before downstream consumers are fixed. Also, unlike the reactive-garbling framework of [NR16], our standard-model construction treats the underlying garbling scheme as a black box and remains compatible with future garbled-circuit advances and optimizations.

Second, we design a bootstrapping mechanism that lifts MHGC to a full-fledged public-key, one-ciphertext, (many-topology) MHFE scheme with simulation-based security. Our bootstrapping step is proved simulation-secure, in contrast to prior bootstrapping-style frameworks of Ananth–Sahai [AS16] and its subsequent adaptations [GKS23, HKM$^+$24, DGKS25a, BKS25], where the security guarantees are tailored to their settings and typically formulated at the indistinguishability level. This simulation guarantee is essential for our goal of characterizing leakage in recursive workflows.

Theorem 1.2 (Informal)**.** *Assuming an adaptively IND-secure public-key FE scheme for* P/poly*, there exists a public-key* MHFE *scheme for* P/poly *satisfying one-ciphertext SIM-security.*

Recall that "one ciphertext" means a single garbled encoding of a database/state carried across hops. As mentioned previously, we note that a one-ciphertext SIM-secure MHFE scheme implies a many-ciphertext IND-secure MHFE scheme by a standard hybrid argument.

1.3 Why Existing Primitives Fall Short

We now revisit existing primitives that might seem to be able to realize the workflows above and where they break down. The use cases require three features simultaneously: (i) computation should proceed on encrypted intermediate state, so expensive work is not repeated, (ii) the workflow must support evolving datasets, in the sense that new inputs encrypted at later times and new analyses authorized at later times extend the ongoing computation rather than triggering a restart, and (iii) intermediate outputs should be released under restricted, role-specific keys. No prior primitive captures all three.

Standard FE and FE for Rich Functionalities. Standard FE decryption returns a deterministic value $f(x)$ [BSW11, O'N10], not a forwardable ciphertext. This prevents further computation on intermediate encrypted states. FE for Turing machines and other rich functionalities [AS16, ACFQ22] expand what can be computed but keep the same single-output interface and follow a single fixed computation path.

Function-Hiding FE: Structural Limitations. Public-key FE is not function-hiding in general [BSW11, O'N10, AGVW13], so a global decryption secret embedded in an FE key would leak to key holders. Function-hiding variants exist only in restricted settings [BRS13a, BRS13b], and even there the output remains a deterministic value rather than a forwardable ciphertext.

Multi-Hop FHE. Multi-Hop FHE [Gen09, GHV10] allows for repeated computation on encrypted data, but any party with the FHE secret key can decrypt every ciphertext, so the key cannot be handed to a user without giving the user access to all plaintexts. To get intermediate outputs, parties would have to query some trusted decryption authority, which would greatly increase the load and communication required by the trusted authority.

Randomized FE. Randomized FE outputs depend on internal randomness [ABF$^+$13, GJKS15, GGHZ16, KSY15, AW17, BKMT21], but the released object is still a value, not a ciphertext bound to an edge and topology. Reinterpreting it as a downstream ciphertext fails because the future wiring is unknown at upstream-key time.

Multi-Input, Multi-Client, and Randomized Multi-Input FE. Multi-input and multi-client FE [GGG+14, CDG+18, ACF+20, CDSG+20] allow one to compute on multiple ciphertexts (which may be from different sources) simultaneously. However, the scheme is single-hop and does not produce a forwardable ciphertext for further computation. While one might consider chaining together a multi-input FE scheme to build MHFE, this runs into several problems (see Section 2.1). Randomized multi-input FE [DGKS25b] is a closer chaining candidate yet still requires known wiring at key issuance.

Decomposable FE and Streaming FE. Decomposable FE [AS17, AM18] releases a single function value once pieces are assembled. Streaming FE [GKS23, DGKS25a, BKS25] accommodates evolving inputs along a single path under a single function key, but not arbitrary DAGs with per-node output policies.

MPC and Non-Interactive MPC. MPC [Yao86, GMW87] avoids a key authority but requires interaction, which is incompatible with our outsourced, asynchronous setting where authorized outputs are requested long after encryption. Non-interactive MPC [HIJ+17, BGI+01] reduces interaction but suffers residual-function leakage and is one-shot rather than allowing for gradual computation across an evolving graph.

Enforcing FHE with FE. Hybrid designs that use FE to open selected FHE ciphertexts hit the function-hiding limitation: the FE key for the opener would contain the FHE secret key, which public-key FE does not protect. iO-based circumventions [JLS21, JLS22] change the assumption profile and still do not directly deliver the clean hop-key syntax with simulation security. We instead bootstrap a secret-key MHGC, never placing a global decryption secret in a hop key.

2 Technical Overview

Our goal is to build a public-key Multi-Hop FE (MHFE) scheme. Before we dive into our constructions, we would like to start with an example topology and go through how Multi-Hop FE would be applied to the example. **Note that the following example is intentionally simple for ease of exposition; the actual primitive can handle much more complex functionality, supporting arbitrary polynomially-sized circuits.**

Example Topology. We now describe how MHFE enables the distributed computation in the figure above. Crucially, MHFE allows the topology to be built *incrementally*: inputs can be encrypted and gates can be added without knowing what future computations will be performed. We illustrate this by walking through the example using a slightly simplified syntax for the MHFE scheme.

- **Setup.** A trusted authority runs the Setup algorithm to obtain the master public key and the master secret key $(\mathsf{mpk}, \mathsf{msk})$ and publishes mpk. Notice that no topology is fixed at this point. The authority only specifies the parameters for the gates and messages.
- **Step 1: Encrypting x_1 and x_2.** Suppose we start with two inputs x_1 and x_2 that we want to compute on, but we *do not know* what computation will be performed yet. Nevertheless, we start by obtaining an encryption key $\mathsf{ek} \leftarrow \mathsf{EncSetup}(\mathsf{mpk})$ and compute:

$$\mathsf{ct}_{e_1} \leftarrow \mathsf{Enc}(\mathsf{ek}, x_1, e_1), \quad \mathsf{ct}_{e_2} \leftarrow \mathsf{Enc}(\mathsf{ek}, x_2, e_2)$$

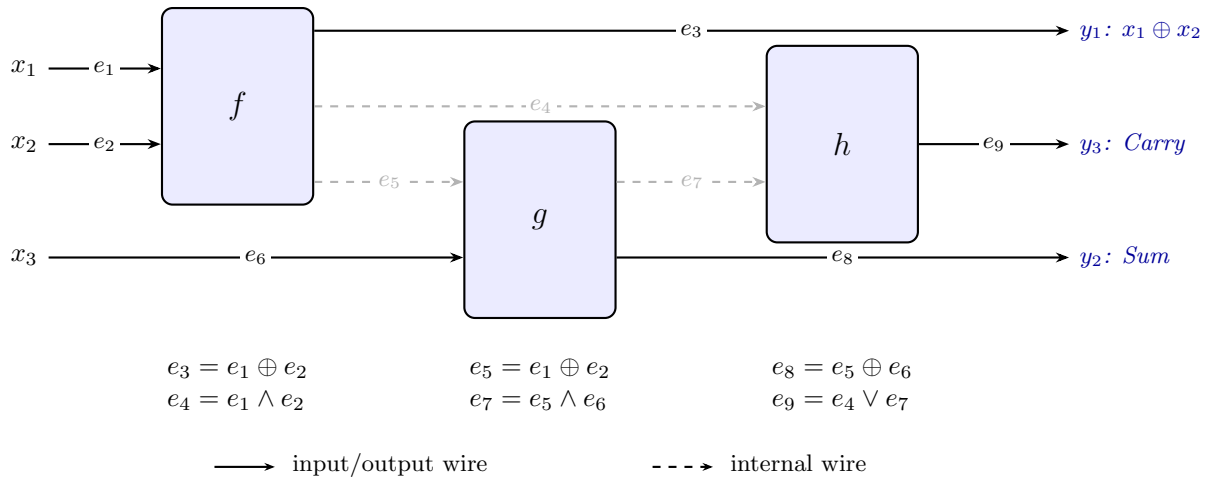

Figure 2: Example topology where $y_3||y_2$ computes the sum of bit wires x_1, x_2, and x_3.

The edge labels e_1, e_2 are fresh identifiers for the two input wires chosen at encryption time. At this point, it is not yet determined what computation will be performed on these ciphertexts.

- **Step 2: Adding gate f.** Later, let's say we decide to compute the function f on e_1 and e_2. Suppose f takes two incoming wires e_1, e_2, and produced three outgoing wires $e_3 = e_1 \oplus e_2$, $e_4 = e_1 \wedge e_2$, and $e_5 = e_1 \oplus e_2$. We would request a function key for gate f from the authority:

$$\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f, \{e_1, e_2\}, \{e_3, e_4, e_5\})$$

  where we specify the incoming wires $\{e_1, e_2\}$ and the outgoing wires $\{e_3, e_4, e_5\}$. We also specify that e_3 is a public output wire while e_4, e_5 are internal wires[3].

  To evaluate the gate, we perform decryption using the function key:

$$(y_1, \mathsf{ct}_{e_4}, \mathsf{ct}_{e_5}) \leftarrow \mathsf{Dec}(\mathsf{sk}_f, \{\mathsf{ct}_{e_1}, \mathsf{ct}_{e_2}\})$$

  This reveals the value on wire e_3 to be $y_1 = x_1 \oplus x_2$ and produces encrypted intermediate values $\mathsf{ct}_{e_4}, \mathsf{ct}_{e_5}$, which can participate in future computations.

- **Step 3: Encrypting x_3 and adding gate g.** Next, we obtain another input x_3 and want to compute a gate g on it together with one of the outputs of f. We first encrypt x_3 using the same encryption key to obtain the ciphertext on wire e_6:

$$\mathsf{ct}_{e_6} \leftarrow \mathsf{Enc}(\mathsf{ek}, x_3, e_6)$$

  We then request a function key for gate g:

$$\mathsf{sk}_g \leftarrow \mathsf{KeyGen}(\mathsf{msk}, g, \{e_5, e_6\}, \{e_7, e_8\})$$

  where g computes an internal wire $e_7 = e_5 \wedge e_6$ and a public output wire $e_8 = e_5 \oplus e_6$. Note that g takes as input the internal wire e_5 from gate f. This particular wiring is specified *now*, not when f was created.

[3]In the formal definition, this is achieved using a type function mapping the edges to their types. Intuitively, one can think about each edge identifier containing a prefix indicating its type: $\mathsf{input}, \mathsf{output}, \mathsf{internal}$.

Lastly, we can evaluate the gate:

$$(\mathsf{ct}_{e_7}, y_2) \leftarrow \mathsf{Dec}(\mathsf{sk}_g, \{\mathsf{ct}_{e_5}, \mathsf{ct}_{e_6}\})$$

This reveals $y_2 = (x_1 \oplus x_2) \oplus x_3$ and gives an intermediate ciphertext ct_{e_7}.

- **Step 4: Adding gate h.** Finally, we compute the carry bit by combining outputs from gates f and g. We request a function key for gate h that computes $e_9 = e_4 \vee e_7$, and then evaluates the gate:

$$\mathsf{sk}_h \leftarrow \mathsf{KeyGen}(\mathsf{msk}, h, \{e_4, e_7\}, \{e_9\})$$

$$y_3 \leftarrow \mathsf{Dec}(\mathsf{sk}_h, \{\mathsf{ct}_{e_4}, \mathsf{ct}_{e_7}\})$$

This reveals $y_3 = (x_1 \wedge x_2) \vee ((x_1 \oplus x_2) \wedge x_3)$ (the carry bit).

At the end of this process, the output wires $y_3||y_2$ give the 2-bit sum of $x_1 + x_2 + x_3$, while the intermediate values on internal wires remain hidden. Recall that the entire topology was built *incrementally*: inputs x_1, x_2 were encrypted before gate f was defined, input x_3 was encrypted right before gate g was defined, and each gate's wiring to future gates was determined only when those future gates were created. This allows for dynamic, evolving computations where the full computation graph does not need to be fixed upfront.

Additionally, the public output wires are also produced in an incremental manner, meaning that one can obtain and make use of partial results as they become available. For example, one can compute y_1 immediately after step 2, before gates g and h are added.

Security Definition. Security states that the entire process given above can be simulated without knowing any of the encrypted wire values. In particular, there exists a PPT simulator Sim where the following process is indistinguishable from the original process:

- **Setup.** The stateful simulator Sim runs the Setup procedure and produces a simulated master public key mpk.

- **Step 1: Encrypting x_1 and x_2.** The simulator produces simulated encryptions for x_1 and x_2 *without* knowing either x_1 or x_2.

$$\mathsf{ct}_{e_1} \leftarrow \mathsf{Sim.Enc}(e_1), \quad \mathsf{ct}_{e_2} \leftarrow \mathsf{Sim.Enc}(e_2)$$

- **Step 2: Adding gate f.** The simulator produces a simulated function key for f. Since the output value $y_1 = e_3 = x_1 \oplus x_2$ would be revealed *in the clear* during normal decryption of f, this value is also given to the simulator. However, the simulator is not given the values of the other output wires e_4 and e_5, since these values would remain hidden during normal decryption.

$$\mathsf{sk}_f \leftarrow \mathsf{Sim.KeyGen}(f, \{e_1, e_2\}, \{e_3, e_4, e_5\}, y_1)$$

Note that since we want the simulated process to be indistinguishable from the original process, then the normal decryption algorithm should still work on the simulated values, i.e.

$$(y_1, \mathsf{ct}_{e_4}, \mathsf{ct}_{e_5}) \leftarrow \mathsf{Dec}(\mathsf{sk}_f, \{\mathsf{ct}_{e_1}, \mathsf{ct}_{e_2}\})$$

where ct_{e_4} and ct_{e_5} are simulated ciphertexts.

- **Step 3: Encrypting x_3 and adding gate g.** Next, the simulator produces a simulated ciphertext for x_3 *without* learning x_3.

$$\mathsf{ct}_{e_6} \leftarrow \mathsf{Sim.Enc}(e_6)$$

The simulator then produces a simulated function key for gate g. Again, the simulator is given the output value $y_2 = e_8$ which would be revealed in the clear, but not the value of e_7.

$$\mathsf{sk}_g \leftarrow \mathsf{Sim.KeyGen}(g, \{e_5, e_6\}, \{e_7, e_8\}, y_2)$$

- **Step 4: Adding gate h.** Finally, the simulator produces a simulated function key for gate h, given the output value $y_3 = e_9$.

$$\mathsf{sk}_h \leftarrow \mathsf{Sim.KeyGen}(h, \{e_4, e_7\}, \{e_9\}, y_3)$$

Thus, we see that the simulator can *incrementally simulate* each of the message and function queries. Note that the only wire values given to the simulator are those that would be revealed *in the clear* during normal decryption. This ensures that all other wire values remain hidden. For example, we would not be able to distinguish between the case where $(x_1, x_2, x_3) = (1, 0, 1)$ and the case where $(x_1, x_2, x_3) = (0, 1, 1)$ since they produce the same output values $(y_1, y_2, y_3) = (1, 0, 1)$, and could thus be simulated by an identical process. Note that any internal wire values, which could also differ, would also remain hidden since they are not given to the simulator.

Multiple Topologies. Our construction also allows for multiple computations to be done on the same input values, allowing reuse of the input ciphertexts. For example, we could later decide to create a function key for a new function p on input wires x_1, x_2, x_3 where $p(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3$. After these initial input values, the computation involving our original topology with (f, g, h) and the computation involving our new topology consisting of just p will operate independently. Both could compute on the same initial input ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3$, but after that, any ciphertexts produced for internal wires would diverge and not be compatible with function keys of other topologies.[4] To specify which topology a function belongs to, we will give a topology identity tid for the function to KeyGen. Note that each topology can be independently and incrementally constructed, and function queries for different topologies can be interwoven.

2.1 Natural Construction Attempts

We first present two natural attempts at constructing MHFE, and illustrate where these attempts run into major challenges.

Chaining Functional Encryption. As a first attempt, we can try using a normal functional encryption scheme FE. A ciphertext for value x_i will simply be an FE ciphertext for x_i. A MHFE function key for f will be a function key for the following function Func_f:

> Func_f(values $\{x_e\}_{e \in [t]}$ corresponding to the input wires to f):
>
> 1. Run $f(\{x_e\}_{e \in [t]})$ to get output values $\{y_\alpha\}_{\alpha \in [n]}$ to be given in the clear and internal wire values $\{x_\beta\}_{\beta \in [m]}$.

[4] To avoid inconsistencies when dealing with multiple topologies, we specify a type function during Setup which maps every wire to a type $\in \{\mathsf{input}, \mathsf{output}, \mathsf{internal}\}$. We require all topologies to share this type matching, so that if a wire is an input wire type in one topology, it is also an input wire type in all topologies.

2. Encrypt each x_β under the FE scheme to get output ciphertext ct_β.

3. Output $(\{y_\alpha\}_{\alpha\in[n]}, \{\mathsf{ct}_\beta\}_{\beta\in[m]})$.

Unfortunately, there are various issues with this attempt. The main issue is that ciphertexts for inputs $\{x_e\}_{e\in[t]}$ to a function may be produced from multiple sources (e.g. as output internal wires from different nodes). Furthermore, these sources may not know which of their outputs will be later used together.

We can try to use multi-input FE [GGG+14] to solve this issue, since multi-input FE allows encrypted values to be produced from separate sources using separate public-keys. However, since our topologies are constructed incrementally, when we produce a function key for Func_f, we may not know which output values of f will be later computed on together. Thus, we will not know to which slot and under which public key we should encrypt each output value x_β. Furthermore, multi-input FE only supports very weak notions of simulation security which will not suffice for our purposes. Even in the IND setting, since Func_f is a randomized function, we would require randomized multi-input FE. However, the only known randomized multi-input FE scheme [DGKS25b] only guarantees indistinguishability when the distributions of the randomized outputs are extremely close, which prevents this chaining from working.

Alternatively, we can try to use decomposable FE [AS17, AM18], which allows for piecewise encryption of a single, larger ciphertext. However, for this to work (and to differentiate it from multi-input FE), each piece of the ciphertext must be encrypted using a shared secret random value for that ciphertext. Again, since we do not know which outputs will be used together in the future, we will not be able to coordinate the usage of these shared random values.

Enforcing FHE Computation Using FE. A second natural attempt is to start from a (multi-hop) FHE scheme, which already supports incremental evaluation on ciphertexts, and use FE as an enforcement layer that selectively "opens" only authorized intermediate values rather than giving out the FHE secret key that enables global decryption. At a high level, the encryptor publishes an FHE public key $\mathsf{pk}_{\mathsf{FHE}}$ (for an associated FHE secret key $\mathsf{sk}_{\mathsf{FHE}}$) and a verification key vk_E for some digital signature scheme, and encrypts each input wire x as $c \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}_{\mathsf{FHE}}, x)$ together with an authenticity tag $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_E, c)$. Independently, the authority publishes a verification key vk_A for a digtial signature scheme together with the master public key $\mathsf{mpk}_{\mathsf{FE}}$ for the underlying FE scheme and, for each hop function f, issues a node token consisting of (i) a certificate $\sigma_f \leftarrow \mathsf{Sign}(\mathsf{sk}_A, f)$ and (ii) an FE key for a verifier–opener circuit that hardwires σ_f and the FHE secret key $\mathsf{sk}_{\mathsf{FHE}}$. Concretely, the MHFE key for a node would include an FE secret key for the following circuit g_{node} (that additionally uses a succinct non-interactive argument (SNARG) system [BCCT12] to attest to correct evaluation on authenticated inputs):

$g_{node}\big((c,\sigma),(c',f,\pi)\big)$:

1. Verify $\mathsf{Verify}(\mathsf{vk}_E, c, \sigma) = 1$.

2. Verify $\mathsf{Verify}(\mathsf{vk}_A, f, \sigma_f) = 1$ (where σ_f is hardwired).

3. Verify the SNARG proof π attesting that $c' = \mathsf{FHE.Eval}(\mathsf{pk}_{\mathsf{FHE}}, f, c)$ for the certified f and signed input ciphertext (c, σ).

4. If all checks pass, output the desired intermediate value by $y \leftarrow \mathsf{FHE.Dec}(\mathsf{sk}_{\mathsf{FHE}}, c')$, and also output an authenticated "next-hop" FHE ciphertext (e.g., c' together with a fresh

certificate under an appropriate signing key hardwired) so that downstream hops can continue to evaluate on c'.

Operationally, a node-holder would first locally compute $c' \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}_{\mathsf{FHE}}, f, c)$, then generate a SNARG proof π that this evaluation was performed correctly for an authority-certified f on an encryptor-signed ciphertext, and finally wrap $\big((c,\sigma),(c',f,\pi)\big)$ inside an FE ciphertext under $\mathsf{mpk}_{\mathsf{FE}}$ and apply its FE key to obtain y and a forwardable authenticated ciphertext. This high level approach seems appealing for MHFE-style asynchronous pipelines: each hop can be processed non-interactively, different node keys do not need to coordinate, and correctness of computation is externally certified by the SNARG while output release is mediated by FE.

Unfortunately, this approach runs into a fundamental obstacle: the FE key for g_{node} must hardwire the FHE decryption key $\mathsf{sk}_{\mathsf{FHE}}$. Standard public-key FE provides no meaningful function privacy by default—the function (and hence any secrets embedded in it) is not hidden from key holders under the standard FE formulations [BSW11, O'N10, AGVW13]. In fact, function-hiding variants of public key FE are known only in restricted settings [BRS13a, BRS13b]. Thus, hardwiring $\mathsf{sk}_{\mathsf{FHE}}$ into g_{node} effectively leaks $\mathsf{sk}_{\mathsf{FHE}}$ to any node key holder, collapsing the intended "authorized output only" guarantee back to global decryption of (multi-hop) FHE. One could try to replace the FE enforcement layer with indistinguishability obfuscation [BGI$^+$01, GGH$^+$13] to hide $\mathsf{sk}_{\mathsf{FHE}}$ inside an obfuscated verifier–decryptor, but this would move us to iO-based techniques and the attendant complexity-leveraging/subexponential-security regime, which we aim to avoid.

2.2 Our Approach

We introduce a different plan of attack: What if we could leverage the power of FE to reduce our problem to a much simpler problem, perhaps one that could even be addressable in Minicrypt? As inspiration, we observe that there exist bootstrapping techniques for FE for Turing Machines and streaming FE [AS16, GKS23] that use FE to lift a secret-key "one-time" scheme to a public-key "many-time" scheme. So we begin by considering what weaker secret-key primitive might be a good starting point for a bootstrapping-based approach.

Looking ahead, we build our scheme in two steps:

1. First, we construct a weaker secret-key primitive which we call a multi-hop garbled circuits (MHGC). We prove the following:

   Theorem 2.1. *Assuming* OWF*s, there exists a* MHGC *scheme for* $\mathsf{P/Poly}$*.*

2. Second, we show how to adapt the bootstrapping techniques from [AS16, GKS23] to build a multi-hop functional encryption scheme (MHFE) from a MHGC scheme. We prove the following:

   Theorem 2.2. *Assuming (1) a secure multi-hop garbled circuit* MHGC*, and (2) an adaptive-IND-secure public-key FE scheme for* $\mathsf{P/Poly}$*, there exists a SIM-secure* MHFE *scheme for* $\mathsf{P/Poly}$*.*

2.2.1 Multi-Hop Garbled Circuits.

We now sketch the secret-key primitive that serves as our starting point: a *multi-hop garbled circuit* scheme MHGC supports incremental, DAG-structured computation where internal wire values remain garbled/encoded and can be forwarded to downstream gates, while designated output wires

are revealed in the clear. A MHGC scheme is essentially an adaptation of the notion of reactive garbling [NR16], a similar primitive which is built in the random oracle model (our construction adapts the construction of [NR16] to the standard model). At a high level, MHGC separates *input garbling* from *circuit garbling* (so inputs can be encoded before the circuit/topology is known), and it enforces safe chaining by binding every wire identifier e to a consistent "encoding interface" derived from a master secret key. The scheme consist of three functions: MHGC.Setup produces a master secret key, MHGC.GarbInp garbles input wires, and MHGC.GarbCirc garbles circuits. Notably, each garbled circuit may also take as input garbled values output by previous circuits, and produce garbled output values for later circuits.

To keep the discussion concrete, we now describe our MHGC construction for the fixed topology in Figure 1 (gates f, g, h over the wires $e_1, \ldots, e_9$). Since the example wires are single bits, we describe the scheme in the input length $\ell_{\mathcal{X}} = 1$ case. The role of MHGC here is to (i) reveal the designated public outputs y_1 on e_3, y_2 on e_8, and y_3 on e_9 in the clear, while (ii) producing *forwardable* encodings for the internal wires e_4, e_5, e_7 so that downstream gates can be evaluated later. Throughout, we use the same type convention as above: e_1, e_2, e_6 are input wires; e_3, e_8, e_9 are output wires; and e_4, e_5, e_7 are internal wires. Our construction will be built from an underlying garbled circuit scheme GC, pseudorandom functions, and secret-key encryption, all of which can be built from OWFs.

- **Setup (one-time, secret key).** The garbler samples two PRF seeds K and K_p and fixes them for the entire computation. These seeds deterministically define, for every wire identifier e:

$$p_e := \mathsf{PRF}_p(K_p, e) \in \{0,1\} \qquad \text{and} \qquad \mathsf{sk}_{e,b} := \mathsf{SKE.Setup}(1^\lambda; \mathsf{PRF}(K, (e,b))) \text{ for } b \in \{0,1\}.$$

  Here p_e is a one-time pad bit for wire e, and $\mathsf{sk}_{e,0}, \mathsf{sk}_{e,1}$ are two symmetric keys associated with the two possible *padded* values on that wire.

- **Step 1: Garbling the input wires e_1, e_2, e_6.** To encode a bit $x_e \in \{0,1\}$ on wire e, the garbler computes the padded bit $\widetilde{x}_e := x_e \oplus p_e$ and outputs the garbled-wire encoding

$$\widehat{X}_e := (\widetilde{x}_e, \mathsf{sk}_{e,\widetilde{x}_e}).$$

  Concretely, we produce $\widehat{X}_{e_1}$ for x_1, $\widehat{X}_{e_2}$ for x_2, and later $\widehat{X}_{e_6}$ for x_3. Notice that $\widehat{X}_e$ contains only one of the two keys $\mathsf{sk}_{e,0}, \mathsf{sk}_{e,1}$, namely the one consistent with the padded bit $\widetilde{x}_e$.

- **Step 2: Garbling and evaluating node f.** Node f consumes the input wires e_1, e_2 and produces a public output e_3 and internal wires e_4, e_5 (where e_5 is an internal copy of e_3 used downstream). The garbler constructs a small *plaintext* circuit P_f that takes $(\widetilde{x}_{e_1}, \widetilde{x}_{e_2})$ as input, unpads via $x_1 = \widetilde{x}_{e_1} \oplus p_{e_1}$ and $x_2 = \widetilde{x}_{e_2} \oplus p_{e_2}$, computes

$$x_{e_3} = x_1 \oplus x_2, \qquad x_{e_4} = x_1 \wedge x_2, \qquad x_{e_5} = x_1 \oplus x_2,$$

  outputs x_{e_3} in the clear, and for each internal output $\beta \in \{e_4, e_5\}$ re-pads and re-encodes it as

$$\widetilde{x}_\beta := x_\beta \oplus p_\beta, \qquad \widehat{X}_\beta := (\widetilde{x}_\beta, \mathsf{sk}_{\beta,\widetilde{x}_\beta}).$$

  Then the garbler garbles P_f using the underlying garbling scheme:

$$(\widetilde{P}_f, \mathsf{lab}_{e_1,0}, \mathsf{lab}_{e_1,1}, \mathsf{lab}_{e_2,0}, \mathsf{lab}_{e_2,1}) \leftarrow \mathsf{GC.Garble}(1^\lambda, P_f),$$

and encrypts each input label under the matching wire-key:

$$\mathsf{ct}_{e_i,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{e_i,b}, \mathsf{lab}_{e_i,b}) \quad \text{for } i \in \{1,2\},\ b \in \{0,1\}.$$

The garbled description of f is $\widehat{F}_f := (\widetilde{P}_f, \mathsf{ct}_{e_1,0}, \mathsf{ct}_{e_1,1}, \mathsf{ct}_{e_2,0}, \mathsf{ct}_{e_2,1})$. To evaluate f, the evaluator uses $\widehat{X}_{e_1} = (\widetilde{x}_{e_1}, \mathsf{sk}_{e_1,\widetilde{x}_{e_1}})$ to decrypt *only* $\mathsf{ct}_{e_1,\widetilde{x}_{e_1}}$ and recover $\mathsf{lab}_{e_1,\widetilde{x}_{e_1}}$ (and similarly for e_2), then obtains

$$(y_1, \widehat{X}_{e_4}, \widehat{X}_{e_5}) \leftarrow \mathsf{GC.Eval}(\widetilde{P}_f, \mathsf{lab}_{e_1,\widetilde{x}_{e_1}}, \mathsf{lab}_{e_2,\widetilde{x}_{e_2}}) \quad \text{where } y_1 = x_{e_3}.$$

Crucially, $\widehat{X}_{e_4}, \widehat{X}_{e_5}$ are now in the *same format* as an input encoding and can be stored and forwarded.

- **Step 3: Garbling and evaluating gate g.** Gate g consumes the internal wire e_5 (produced by f) and the fresh input wire e_6, and produces an internal wire e_7 and public output e_8 (the sum bit). The garbler repeats the same template: build a plaintext circuit P_g that unpads the two inputs using p_{e_5}, p_{e_6}, computes $x_{e_7} = x_{e_5} \wedge x_{e_6}$ and $x_{e_8} = x_{e_5} \oplus x_{e_6}$, outputs x_{e_8} in the clear, and outputs $\widehat{X}_{e_7} = (x_{e_7} \oplus p_{e_7}, \mathsf{sk}_{e_7,x_{e_7}\oplus p_{e_7}})$ for the internal wire. The garbler garbles P_g to get labels for e_5, e_6, encrypts those labels under $\mathsf{sk}_{e_5,b}$ and $\mathsf{sk}_{e_6,b}$, and publishes the resulting $\widehat{F}_g$. Evaluation uses the already-available encodings $\widehat{X}_{e_5}$ (from f) and $\widehat{X}_{e_6}$ (from garbling x_3) to decrypt exactly one label per input wire and then runs $\mathsf{GC.Eval}$:

$$(\widehat{X}_{e_7}, y_2) \quad \text{where } y_2 = x_{e_8}.$$

- **Step 4: Garbling and evaluating gate h.** Gate h consumes the internal wires e_4 and e_7 and outputs the public wire e_9 (the carry bit). Again we build a plaintext circuit P_h that unpads e_4, e_7, computes $x_{e_9} = x_{e_4} \vee x_{e_7}$, and outputs it in the clear; garble it to obtain labels for e_4, e_7; encrypt those labels under $\mathsf{sk}_{e_4,b}$ and $\mathsf{sk}_{e_7,b}$; and publish $\widehat{F}_h$. Evaluation uses $\widehat{X}_{e_4}$ and $\widehat{X}_{e_7}$ to recover the correct labels and runs $\mathsf{GC.Eval}$, yielding $y_3 = x_{e_9}$ in the clear.

The only reason g can later consume e_5 (and h can later consume e_4, e_7) is that every internal output wire is re-encoded as $\widehat{X}_e = (\widetilde{x}_e, \mathsf{sk}_{e,\widetilde{x}_e})$ using the globally consistent, PRF-derived $(p_e, \mathsf{sk}_{e,0}, \mathsf{sk}_{e,1})$. Thus, outputs of f are already well-formed inputs for g and h when those gates are garbled/evaluated later, matching the incremental/topological workflow in Figure 1.

Proof sketch for the example topology. *Correctness* follows by evaluating gates in the natural topological order ($f \to g \to h$): at each gate, the evaluator can decrypt exactly the two needed garbling labels (one for each input wire) because $\widehat{X}_e$ contains precisely the key $\mathsf{sk}_{e,\widetilde{x}_e}$ matching the padded bit, and then $\mathsf{GC.Eval}$ correctly computes the plaintext circuit P_f, P_g, P_h, which explicitly unpads, applies the intended Boolean gate, and re-encodes internal outputs for the next hop.

Security (simulation-style) can be seen by progressively replacing, gate-by-gate in the same topological order, each real garbled gate $\widehat{F}_f, \widehat{F}_g, \widehat{F}_h$ with a simulated garbling of the corresponding P_f, P_g, P_h that is consistent only with the *revealed* public outputs (y_1, y_2, y_3): (1) PRF security lets us treat the per-wire pads/keys as pseudorandom; (2) since each input wire encoding reveals only one of $\mathsf{sk}_{e,0}, \mathsf{sk}_{e,1}$, semantic security of SKE lets us hide the "unused" encrypted label at each gate (the label under the key the evaluator does not possess); and (3) once the unused labels are information-theoretically irrelevant to evaluation, standard garbling privacy lets us replace the garbled circuits with simulations given only the gate outputs (and for internal wires, the simulator can simply sample fresh-looking encodings $\widehat{X}_{e_4}, \widehat{X}_{e_5}, \widehat{X}_{e_7}$ and then reuse them consistently as inputs

to downstream simulated gates). For this fixed topology, the simulator therefore generates a view indistinguishable from real using only (y_1, y_2, y_3), which is exactly the intended leakage for the example workflow.

Our multi-hop garbled circuit scheme is essentially a secret-key version of MHFE that only supports function keys for a single topology.

2.2.2 Bootstrapping.

We now show how to bootstrap a MHGC scheme to a MHFE scheme. The core idea is that we will use a public-key FE scheme to securely generate at decryption time, fresh MHGC master secret keys for each topology. The output of this FE decryption will then include primitives which both (1) hide the MHGC master secret key, and (2) allow the user to construct garbled inputs and garbled circuits for the MHGC scheme under the generated MHGC master secret key. We can then rely on the underlying MHGC scheme for correctness and security of our scheme.

Our construction utilizes two functional encryption schemes alongside MHGC:

- MHGC: a multi-hop garbled circuit scheme. Provided with a master secret key $\mathsf{MHGC.msk}$, we can garble inputs via $\mathsf{MHGC.GarbInp}(\mathsf{MHGC.msk}, x, e)$ to produce garbled inputs $\widehat{X}_e$, and garble circuits via $\mathsf{MHGC.GarbCirc}(\mathsf{MHGC.msk}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$ to produce garbled circuits $\widehat{F}_v$. Evaluation $\mathsf{MHGC.Eval}(\widehat{F}_v, \{\widehat{X}_e\}_{e \in S_{\mathsf{in}}})$ produces plaintext outputs and new garbled inputs for downstream gates.

- FE: a public-key FE scheme. This scheme will be responsible for generating a fresh MHGC master secret key $\mathsf{MHGC.msk}_{\mathsf{tid}}$ and corresponding garbled circuits $\widehat{F}_{\mathsf{tid},v}$ for each topology tid.

- FPFE: a function-private, secret-key FE scheme.[5] This scheme will use the MHGC master secret key generated by FE to garble each input value x_e under MHGC.

To perform the bootstrapping, we will perform the following process. Note, that this construction is also depicted in the following diagram. In our construction, it is important to distinguish between (1) incoming edges from input wires, (2) incoming edges from internal wires, (3) outgoing edges to output wires, and (4) outgoing edges to internal wires. We use $S^{\mathsf{inp}}_{\mathsf{in}}, S^{\mathsf{int}}_{\mathsf{in}}, S^{\mathsf{out}}_{\mathsf{out}}, S^{\mathsf{int}}_{\mathsf{out}}$ respectively to denote these four classes of wires.[6]

- **Setup:** Generate FE keys $(\mathsf{FE.mpk}, \mathsf{FE.msk})$ which serve as the master public key and master secret key of our MHFE scheme.

- **Encryption Setup:** Generate an FPFE master secret key $\mathsf{FPFE.msk}$ and PRF keys $\mathsf{PRF}_1.K$, $\mathsf{PRF}_2.K$. Create an FE ciphertext $\mathsf{FE.ct}$ that encrypts $(\mathsf{FPFE.msk}, \mathsf{PRF}_1.K, \mathsf{PRF}_2.K)$. The encryption key is $(\mathsf{FPFE.msk}, \mathsf{FE.ct})$.

- **Encryption:** To encrypt a value x_e on edge e, create an FPFE ciphertext $\mathsf{FPFE.ct}_e$ encrypting (x_e, e). The MHFE ciphertext is $\mathsf{CT}_e = (\mathsf{FE.ct}, \mathsf{FPFE.ct}_e)$.

- **Key Generation:** To generate a function key for gate f at vertex v in topology tid with input edges S_{in} and output edges S_{out}, create an FE function key $\mathsf{FE.sk}_{G_{\mathsf{tid},v}}$ for the function

[5] 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].

[6] For ease of explanation, we have also 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.

$G_{\mathsf{tid},v}$ which takes as input $(\mathsf{FPFE.msk}, \mathsf{PRF}_1.K, \mathsf{PRF}_2.K)$ and does the following. We will use the PRF keys to generate any needed randomness, where $\mathsf{PRF}_1.K$ will generate randomness shared by all function in the topology and $\mathsf{PRF}_2.K$ will generate randomness unique to each vertex in the topology:

1. Generate a fresh $\mathsf{MHGC.msk}_{\mathsf{tid}}$ and a new PRF key $\mathsf{PRF}_3.K_{\mathsf{tid}}$ for topology tid.
2. Construct the function $H_{\mathsf{tid},v}$ hardwired with $(\mathsf{MHGC.msk}_{\mathsf{tid}}, \mathsf{PRF}_3.K_{\mathsf{tid}})$. Given input (x_e, e), the function $H_{\mathsf{tid},v}$ uses $\mathsf{PRF}_3.K_{\mathsf{tid}}$ for randomness and it outputs the garbled inputs $\widehat{X}_{\mathsf{tid},e} \leftarrow \mathsf{MHGC.GarbInp}(\mathsf{MHGC.msk}_{\mathsf{tid}}, x_e, e)$.
3. Use $\mathsf{FPFE.msk}$ (from the FE ciphertext) to generate an FPFE function key $\mathsf{FPFE.sk}_{H_{\mathsf{tid},v}}$ for $H_{\mathsf{tid},v}$.
4. Generate the garbled circuit $\widehat{F}_{\mathsf{tid},v} \leftarrow \mathsf{MHGC.GarbCirc}(\mathsf{MHGC.msk}_{\mathsf{tid}}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$.
5. Output $\mathsf{SK}_{\mathsf{tid},v} = (\widehat{F}_{\mathsf{tid},v}, \mathsf{FPFE.sk}_{H_{\mathsf{tid},v}})$.

- **Decryption:** To decrypt input ciphertexts $\{\mathsf{CT}_e\}_{e \in S_{\mathsf{in}}}$ using function key $\mathsf{SK}_{\mathsf{tid},v}$:

1. Use FE to decrypt $\mathsf{FE.ct}$ with $\mathsf{FE.sk}_{G_{\mathsf{tid},v}}$ to obtain $(\widehat{F}_{\mathsf{tid},v}, \mathsf{FPFE.sk}_{H_{\mathsf{tid},v}})$.
2. For each incoming edge from an input wire $e \in S^{\mathsf{inp}}_{\mathsf{in}}$, use FPFE to decrypt $\mathsf{FPFE.ct}_e$ with $\mathsf{FPFE.sk}_{H_{\mathsf{tid},v}}$ to obtain garbled input $\widehat{X}_{\mathsf{tid},e}$.
3. For each incoming edge $e \in S^{\mathsf{int}}_{\mathsf{in}}$ from an internal wire, recover $\widehat{X}_{\mathsf{tid},e}$ from the ciphertext CT_e.
4. Use $\mathsf{MHGC.Eval}(\widehat{F}_{\mathsf{tid},v}, \{\widehat{X}_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}})$ to obtain the values $\{x_\alpha\}_{\alpha \in S^{\mathsf{out}}_{\mathsf{out}}}$ for output edges, and the garbled outputs $\{x_\beta\}_{\beta \in S^{\mathsf{int}}_{\mathsf{out}}}$ for internal edges.

Correctness follows from the correctness of the underlying schemes. For security, we focus on each topology at a time. In the hybrid proof, on a very high level, we first hardwire the outputs of $G_{\mathsf{tid},v}$ (the garbled circuit $\widehat{F}_{\mathsf{tid},v}$ and FPFE function key) into an SKE ciphertext inside the FE function key, with the corresponding SKE secret key placed inside the FE ciphertext. This allows us to remove the values $\mathsf{FPFE.msk}$ and PRF keys from the FE ciphertext via FE security. Next, we hardwire the garbled inputs $\widehat{X}_{\mathsf{tid},e}$ inside the FPFE function keys, allowing us to use FPFE security to remove the input values from the FPFE ciphertexts. Finally, we use function privacy of FPFE to remove the MHGC master secret key from the FPFE function keys, and then apply MHGC security to complete the proof.

Adapting the bootstrapping framework to MHFE presents several challenges beyond the streaming FE setting. First, MHFE supports a DAG structure rather than a linear stream: the outputs of one gate can feed into multiple downstream gates, and a gate can receive inputs from multiple upstream gates. This means our MHGC scheme must produce "re-encryptable" outputs–garbled values that can serve as inputs to subsequent garbled circuits. Second, the topology is built *incrementally*: inputs are encrypted before the computation graph is known, and the wiring between gates is specified only when function keys are requested. This requires careful coordination between the FE and FPFE layers to ensure that garbled inputs produced by H are compatible with garbled circuits produced by G, even though they are generated at different times. Finally, our security proof requires queries to arrive in topological order, ensuring that all input values to a gate are known before its function key is generated. This constraint is essential for the hybrid argument where we hardwire garbled inputs into the FPFE function keys.

We remark that prior work [AS16, GKS23] used bootstrapping to produce an IND-secure scheme. Our work adapts the bootstrapping to support SIM-security. This requires extra steps for hardwiring each of the output values.

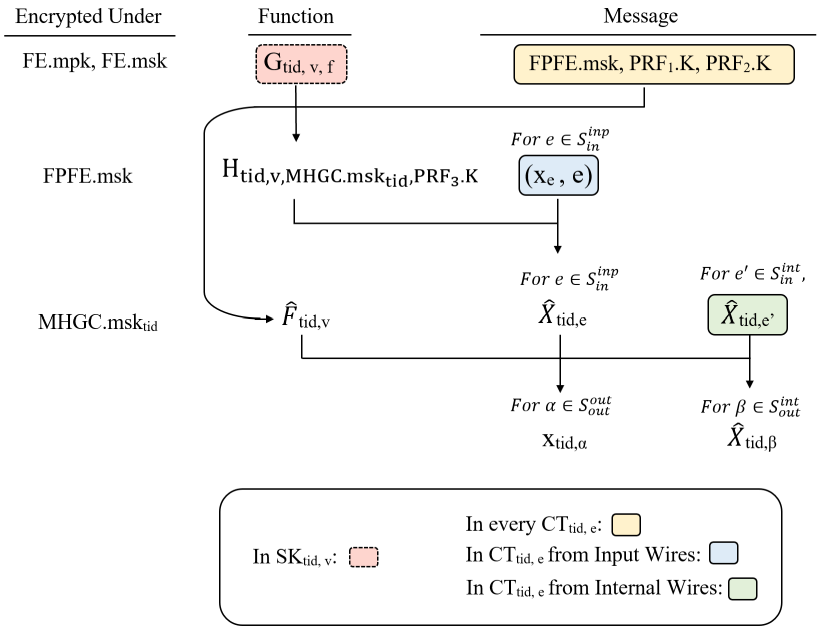

Figure 3: Bootstrapping to a MHFE scheme. Here, we depict the computation regarding a function f on vertex v of topology tid, where $f(\{x_e\}_{e\in S_{\mathsf{in}}^{\mathsf{inp}}}, \{x_{\mathsf{tid},e'}\}_{e'\in S_{\mathsf{in}}^{\mathsf{int}}}) = (\{x_{\mathsf{tid},\alpha}\}_{\alpha\in S_{\mathsf{out}}^{\mathsf{out}}}, \{x_{\mathsf{tid},\beta}\}_{\beta\in S_{\mathsf{out}}^{\mathsf{int}}})$.

3 Preliminaries

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

Notation.

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

3.1 Standard Notions

We use the standard definitions of one way functions (OWFs), pseudorandom functions (PRFs), secret-key encryption (SKE) with pseudorandom ciphertexts, and garbled circuits (GC). We formally define the latter three below.

Definition 3.1 (Pseudorandom Function (PRF))**.** *A pseudorandom function family (PRF) with key space $\mathcal{K} = \{\mathcal{K}_{\lambda,n,m}\}_{\lambda,n,m\in\mathbb{N}}$ is a tuple of PPT algorithms* $\mathsf{PRF} = (\mathsf{PRF.Setup}, \mathsf{PRF.Eval})$ *where*

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

- $\mathsf{PRF.Eval}(K, x)$ *is a deterministic algorithm that takes as input a key* $K \in \mathcal{K}_{\lambda,n,m}$ *and an input* $x \in \{0,1\}^n$*, and outputs a value* $y \in \{0,1\}^m$*.*

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

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

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

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

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

Definition 3.2 (Garbling Scheme)**.** *A garbling scheme is a tuple of* PPT *algorithms* $\mathsf{GC} = (\mathsf{Garble}, \mathsf{Eval})$ *defined as follows:*

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

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

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

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

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

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

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

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

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

Lemma 3.3 ([Yao86])**.** *If there exist* OWF*s, then there exists a garbling scheme.*

Definition 3.4 (Secret Key Encryption (SKE))**.** *A secret key encryption scheme with key space $\mathcal{K} = \{\mathcal{K}_\lambda\}_\lambda$ and ciphertext size $m(\cdot)$ is a tuple of PPT algorithms $\mathsf{SKE} = (\mathsf{SKE.Setup}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ where*

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

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

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

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

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

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

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

1. ***Parameters**: $\mathcal{A}$ takes as input 1^λ.*
2. ***Setup**: $\mathsf{sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$.*
3. ***Challenge Message Queries**: The following can be repeated any polynomial number of times:*
   (a) *$\mathcal{A}$ outputs a challenge message pair (x_0, x_1) where $|x_0| = |x_1|$.*
   (b) *$\mathsf{ct}_b \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}, x_b)$*
   (c) *Sent ct_b to $\mathcal{A}$.*

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

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

Definition 3.5 (Secret Key Encryption with Pseudorandom Ciphertexts)**.** *A secret key encryption scheme with key space $\mathcal{K} = \{\mathcal{K}_{\lambda,n}\}_{\lambda,n\in\mathbb{N}}$ and ciphertext size $m(\cdot)$ has pseudorandom ciphertexts if there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$ and every PPT adversary $\mathcal{A}$,*

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

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

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

1. ***Parameters***: *$\mathcal{A}$ takes as input 1^λ.*
2. ***Setup***: $\mathsf{sk} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$
3. ***Challenge Message Queries***: *The following can be repeated any polynomial number of times:*
   (a) *$\mathcal{A}$ outputs a challenge message x where $x \in \{0,1\}^*$.*
   (b) *If $b = 0$,* $\mathsf{ct} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}, x)$.
   (c) *If $b = 1$,* $\mathsf{ct} \leftarrow \{0,1\}^{m(\lambda,|x|)}$
   (d) *Send* ct *to $\mathcal{A}$.*
4. ***Experiment Outcome***: *$\mathcal{A}$ outputs a bit b' which is the output of the experiment.*

3.2 Public-Key Functional Encryption

Definition 3.6 (Public-Key Functional Encryption)**.** *A public-key functional encryption scheme for* $\mathsf{P/Poly}$ *is a tuple of PPT algorithms* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ *defined as follows:*

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

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

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

We now define adaptive security.

Definition 3.7 (Adaptive-IND Security for Public-Key FE)**.** *A public-key functional encryption scheme* FE *for* $\mathsf{P/Poly}$ *is adaptive-IND secure if there exists a negligible function* μ *such that for all* $\lambda \in \mathbb{N}$ *and every PPT adversary* $\mathcal{A}$*,*

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

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

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

1. ***Parameters****:* $\mathcal{A}$ *takes as input* 1^λ*, and outputs a function size* $1^{\ell_\mathcal{F}}$*, an input size* $1^{\ell_\mathcal{X}}$*, and an output size* $1^{\ell_\mathcal{Y}}$*.*
2. ***Setup****: Compute* $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}})$*.*
3. ***Public Key****: Send* mpk *to* $\mathcal{A}$*.*
4. *The adversary can make a polynomial number of function queries and message queries in any order:*
   - ***Function Query****:*
     (a) $\mathcal{A}$ *outputs a function query* $f \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$
     (b) $\mathsf{sk}_f \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}, f)$
     (c) Send sk_f *to* $\mathcal{A}$
   - ***Message Query****:*
     (a) $\mathcal{A}$ *outputs a challenge message pair* (x_0, x_1) *where* $x_0, x_1 \in \{0,1\}^{\ell_\mathcal{X}}$*.*
     (b) $\mathsf{ct} \leftarrow \mathsf{FE.Enc}(\mathsf{mpk}, x_b)$
     (c) Send ct *to* $\mathcal{A}$*.*
5. ***Experiment Outcome****:* $\mathcal{A}$ *outputs a bit* b' *which is the output of the experiment.*

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

3.3 Function-Private Secret-Key Functional Encryption

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

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.*

A function-private FE scheme also achieves the following security definition:

Definition 3.9 (Function-Private Adaptive-IND Security for Secret-Key FE)**.** *A secret-key functional encryption scheme* FE *for* $\mathsf{P/Poly}$ *is function-private adaptive-IND secure if there exists a negligible function* μ *such that for all* $\lambda \in \mathbb{N}$ *and every PPT adversary* $\mathcal{A}$*,*

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

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

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

1. ***Parameters**: $\mathcal{A}$ takes as input 1^λ, and outputs a function size $1^{\ell_\mathcal{F}}$, an input size $1^{\ell_\mathcal{X}}$, and an output size $1^{\ell_\mathcal{Y}}$.*
2. ***Setup**: $\mathsf{msk} \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{X}}, 1^{\ell_\mathcal{Y}})$.*
3. *The adversary can make a polynomial number of function queries and message queries in any order:*
   - ***Function Query**:*
     (a) $\mathcal{A}$ outputs a function query pair (f_0, f_1) where $f_0, f_1 \in \mathcal{F}[\ell_\mathcal{F}, \ell_\mathcal{X}, \ell_\mathcal{Y}]$
     (b) $\mathsf{sk}_f \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}, f_b)$
     (c) Send sk_f to $\mathcal{A}$
   - ***Message Query**:*
     (a) $\mathcal{A}$ outputs a challenge message pair (x_0, x_1) where $x_0, x_1 \in \{0,1\}^{\ell_\mathcal{X}}$.
     (b) $\mathsf{ct} \leftarrow \mathsf{FE.Enc}(\mathsf{msk}, x_b)$
     (c) Send ct to $\mathcal{A}$.
4. ***Experiment Outcome**: $\mathcal{A}$ outputs a bit b' which is the output of the experiment.*

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

4 Definitions

4.1 Multi-Hop Functional Encryption (MHFE)

We define our main result, a multi-hop functional encryption scheme.

Definition 4.1 (Multi-Hop Functional Encryption)**.** *A public-key Multi-Hop Functional Encryption scheme (MHFE) for $\mathsf{P/Poly}$ is a tuple of Probabilistic Polynomial Time (PPT) algorithms $\mathsf{MHFE} = (\mathsf{MHFE.Setup}, \mathsf{MHFE.EncSetup}, \mathsf{MHFE.Enc}, \mathsf{MHFE.KeyGen}, \mathsf{MHFE.Dec})$ defined as follows:*

- $\mathsf{MHFE.Setup}(1^\lambda, 1^{d_\mathsf{max}}, 1^{\ell_\mathcal{F}}, 1^{\ell_\mathcal{X}}, \mathsf{type})$*: takes as input the security parameter λ, a maximum degree d_max, a function description size $\ell_\mathcal{F}$, a message size $\ell_\mathcal{X}$, and a function $\mathsf{type} : \mathcal{E} \to \{\mathsf{input}, \mathsf{output}, \mathsf{internal}\}$ mapping edges to edge types, where*
  - *$\mathcal{F}$ is the function space containing all circuits with descriptions in $\{0,1\}^{\ell_\mathcal{F}}$,*
  - *$\mathcal{X} = \{0,1\}^{\ell_\mathcal{X}}$ is the message space,*
  - *$\mathcal{V} = \{0,1\}^\lambda$ is the vertex space,*
  - *$\mathcal{E} = \{0,1\}^\lambda$ is the edge space, with $\mathcal{E}^\mathsf{inp}, \mathcal{E}^\mathsf{out}, \mathcal{E}^\mathsf{int}$ being the restrictions of $\mathcal{E}$ to edges with type $\mathsf{input}, \mathsf{output}, \mathsf{internal}$ respectively.*

  and outputs the master public key mpk and the master secret key msk.

- $\mathsf{MHFE.EncSetup}(\mathsf{mpk})$*: takes as input the master public key* mpk *and outputs an encryption key* ek*.*
- $\mathsf{MHFE.Enc}(\mathsf{ek}, x, e)$*: takes as input an encryption key* ek*, a message* $x \in \mathcal{X}$*, and an edge* $e \in \mathcal{E}^{\mathsf{inp}}$*, and outputs an encryption* ct_e *of* x*.*
- $\mathsf{MHFE.KeyGen}(\mathsf{msk}, \mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$*: takes as input the master secret key* msk*, a topology id* $\mathsf{tid} \in \{0,1\}^\lambda$*, a function* $f \in \mathcal{F}$ *with* $f : \mathcal{X}^{|S_{\mathsf{in}}|} \to \mathcal{X}^{|S_{\mathsf{out}}|}$*, a vertex label* $v \in \mathcal{V}$*, and an ordered set of incoming edges* $S_{\mathsf{in}} \subseteq \mathcal{E}^{\mathsf{inp}} \cup \mathcal{E}^{\mathsf{int}}$ *and outgoing edges* $S_{\mathsf{out}} \subseteq \mathcal{E}^{\mathsf{out}} \cup \mathcal{E}^{\mathsf{int}}$ *with* $|S_{\mathsf{in}}|, |S_{\mathsf{out}}| \leq d_{\mathsf{max}}$*, and outputs a function key* $\mathsf{sk}_{\mathsf{tid},f,v,S_{\mathsf{in}},S_{\mathsf{out}}}$*.*
- $\mathsf{MHFE.Dec}(\mathsf{sk}_{\mathsf{tid},f,v,S_{\mathsf{in}},S_{\mathsf{out}}}, \{\mathsf{ct}_e\}_{e\in S_{\mathsf{in}}})$*: takes as input a function key* $\mathsf{sk}_{\mathsf{tid},f,v,S_{\mathsf{in}},S_{\mathsf{out}}}$ *and ciphertexts* $\{\mathsf{ct}_e\}_{e\in S_{\mathsf{in}}}$ *corresponding to the incoming edges to vertex* v*, and outputs values* $(\{x_\alpha\}_{\alpha\in S^{\mathsf{out}}_{\mathsf{out}}}, \{\mathsf{ct}_\beta\}_{\beta\in S^{\mathsf{int}}_{\mathsf{out}}})$ *corresponding to the outgoing edges of vertex* v*, where* $S^{\mathsf{out}}_{\mathsf{out}}$ *and* $S^{\mathsf{int}}_{\mathsf{out}}$ *are the restrictions of* S_{out} *to edges of type* output *and* $\mathsf{internal}$ *respectively.*

MHFE *schemes must satisfy correctness (Definition 4.8) and security (Definition 4.10) which we will define later.*

While the notions of correctness and security for MHFE are quite intuitive, the formal definitions can be complex. We refer the reader to the technical overview for an intuitive explanation.

At a high level, for each topology tid, we consider the function queries made to an MHFE scheme as implicitly forming a directed acyclic multigraph, where for each function query, the vertex v represents a computation unit, labeled with a corresponding bounded-size circuit f, and a set of input wires S_{in} and output wires S_{out}. The input and output wires correspond to edges $e \in \mathcal{E}$ in the graph, which allow data to flow between computation units. Intuitively, an edge from vertex u to vertex v represents that the output of the computation unit u is fed into the computation unit v as input. Message queries correspond to encrypting data onto incoming edges (to all topologies) of our computation units.

Informally, correctness requires that for each topology, if you were to trace the data from the message queries through the graph formed by the function queries, then decryption produces the correct outputs which would appear on the corresponding output wires of the graph. Informally, security requires that the entire process should be simulatable without knowing any of the unrevealed encrypted wire values.

4.1.1 Helper Definitions.

To formally define correctness and security, we must introduce additional notions. We start with some notation.

Notation. Let S be any set of edges. If we have an edge labeling function $\mathsf{type} : S \to \{\mathsf{input}, \mathsf{output}, \mathsf{internal}\}$, then we define $S^{\mathsf{inp}}, S^{\mathsf{out}}, S^{\mathsf{int}}$ to be the restrictions of S to edges with type $\mathsf{input}, \mathsf{output}, \mathsf{internal}$ respectively, i.e. $S^{\mathsf{inp}} := \{e \in S : \mathsf{type}(e) = \mathsf{input}\}$, $S^{\mathsf{out}} := \{e \in S : \mathsf{type}(e) = \mathsf{output}\}$, $S^{\mathsf{int}} := \{e \in S : \mathsf{type}(e) = \mathsf{internal}\}$.

We now define a valid query set as a set of message and function queries (for a single topology) which produces a well-defined graph for our computation.

Definition 4.2 (Valid Query Set)**.** *Consider a set*

$$W = (\{(x_i, e_i)\}_{i\in[n]}, \{(\mathsf{tid}, f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j\in[m]})$$

of encryption and keygen queries to a MHFE *scheme for a single topology* tid*, where* $\{(x_i, e_i)\}_{i\in[n]}$ *are the queries to* MHFE.Enc*, and* $\{(\mathsf{tid}, f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j\in[m]}$ *are the queries to* MHFE.KeyGen*. We say that this query set is a* ***valid query set*** *with parameters* $(\lambda, d_{\mathsf{max}}, \ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \mathsf{type})$ *if the following are satisfied:*

- ***Correct Parameters and Edge Types:*** *Let* $\mathcal{F}$ *be the set of functions with descriptions in* $\{0,1\}^{\ell_{\mathcal{F}}}$*,* $\mathcal{X} = \{0,1\}^{\ell_{\mathcal{X}}}$*,* $\mathcal{V} = \{0,1\}^{\lambda}$*, and* $\mathcal{E} = \{0,1\}^{\lambda}$*. Then,*
  - $\forall i \in [n]$*,* $e_i \in \mathcal{E}^{\mathsf{inp}}$ *and* $x_i \in \mathcal{X}$*.*
  - $\forall j \in [m]$*,* $v_j \in \mathcal{V}$ *and* $f_j \in \mathcal{F}$ *is a circuit* $f_j : \mathcal{X}^{|S_{\mathsf{in},j}|} \to \mathcal{X}^{|S_{\mathsf{out},j}|}$*.*
  - $\forall j \in [m]$*,* $S_{\mathsf{in},j} \subseteq \mathcal{E}^{\mathsf{inp}} \cup \mathcal{E}^{\mathsf{int}}$ *and* $S_{\mathsf{out},j} \subseteq \mathcal{E}^{\mathsf{out}} \cup \mathcal{E}^{\mathsf{int}}$ *and* $|S_{\mathsf{in},j}|, |S_{\mathsf{out},j}| \leq d_{\mathsf{max}}$
- ***Unique Encryption Edges:*** *If* $i \neq i'$*, then* $e_i \neq e_{i'}$*.*
- ***Unique Vertices****: If* $j \neq j'$*, then* $v_j \neq v_{j'}$*.*
- ***No Hyperedges:*** *If* $j \neq j'$*,* $S_{\mathsf{in},j} \cap S_{\mathsf{in},j'} = \emptyset$ *and* $S_{\mathsf{out},j} \cap S_{\mathsf{out},j'} = \emptyset$*.*
- ***No Cycles****: There does not exist* $k \geq 1$ *and a set* $I = \{\mathsf{ind}_1, \ldots, \mathsf{ind}_k\} \subseteq [m]$ *such that*
  1. $\forall j \in [k-1]$*,* $|S_{\mathsf{out},\mathsf{ind}_j} \cap S_{\mathsf{in},\mathsf{ind}_{j+1}}| > 0$*.*
  2. $|S_{\mathsf{out},\mathsf{ind}_k} \cap S_{\mathsf{in},\mathsf{ind}_1}| > 0$*.*

Remark 4.3. The requirement that our graph will have no hyperedges does not restrict the usability of our scheme. For example, if a user wants to compute on wire e but also expects that this value will be needed later, then the user could simply first run e through a function that creates copies of the value of wire e and then use one of these copies. In this way, other copies of this value will remain for later use.

Next, we define a *distributed computation transcript*, which describes an instance of data flowing through a directed graph, where each vertex is labeled with the function it computes, and each edge is labeled with a data value.

Definition 4.4 (Distributed Computation Transcript)**.** *A distributed computation transcript is a tuple* $\pi = (G, \mathcal{F}, \mathcal{X}, \mathsf{code}, \mathsf{data})$ *where*

- $G = (V, E, \mathsf{source}, \mathsf{target})$ *is a directed acyclic multigraph*[7] *where*
  - V *is a set of vertices,* E *is a set of edges,*
  - $\mathsf{source} : E \to V$ *is a function assigning to each edge its source vertex,*
  - $\mathsf{target} : E \to V$ *is a function assigning to each edge its target vertex.*
- $\mathcal{F}$ *is the function space and* $\mathcal{X}$ *is the data space,*
- $\mathsf{code} : V \to \mathcal{F} \cup \{\bot\}$ *is a function assigning to each vertex* $v \in V$ *a circuit* $f \in \mathcal{F} \cup \{\bot\}$*,*
- $\mathsf{data} : E \to \mathcal{X} \cup \{\bot\}$ *is a function assigning to each edge* $e \in E$ *a data value* $x \in \mathcal{X} \cup \{\bot\}$*.*

[7] A directed acyclic multigraph is similar to a directed acyclic graph (DAG) except that we allow multiple/parallel edges between any pair of nodes. Thus, rather than labeling an edge from u to v as (u, v), which could result in duplicate and ambiguous edge labels, we give each such edge a unique label e and define $\mathsf{source}(e) = u$ and $\mathsf{target}(e) = v$.

We now define the induced transcript of a valid query set to be the distributed computation transcript formed by tracing the data flow from the message queries through the graph formed by the function queries. To ensure the graph is well-defined, we attach dangling edges of the graph to corresponding source or sink nodes.

Definition 4.5 (Induced Transcript of a Valid Query Set)**.** *Let*

$$W = (\{(x_i, e_i)\}_{i\in[n]}, \{(\mathsf{tid}, f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j\in[m]})$$

be a valid query set with parameters $(\lambda, d_{\mathsf{max}}, \ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \mathsf{type})$*. The* ***induced transcript*** *of* W *is the distributed computation transcript*

$$\pi = (G, \mathcal{F}, \mathcal{X}, \mathsf{code}, \mathsf{data})$$

where

- *We define* $G = (V, E, \mathsf{source}, \mathsf{target})$ *where*
  - $V = \{v_j\}_{j\in[m]} \cup \{v_{\mathsf{source}}, v_{\mathsf{sink}}\}$.
  - $E = \{e_i\}_{i\in[n]} \cup \left(\bigcup_{j\in[m]}(S_{\mathsf{in},j} \cup S_{\mathsf{out},j})\right)$
  - $\mathsf{source}(e) = \begin{cases} v_j & \text{if } e \in S_{\mathsf{out},j} \text{ for exactly one } j \in [m] \\ v_{\mathsf{source}} & \text{else} \end{cases}$
  - $\mathsf{target}(e) = \begin{cases} v_j & \text{if } e \in S_{\mathsf{in},j} \text{ for exactly one } j \in [m] \\ v_{\mathsf{sink}} & \text{else} \end{cases}$
- *We define* $\mathcal{F}$ *as the set of all functions with descriptions in* $\{0,1\}^{\ell_{\mathcal{F}}}$*, and define* $\mathcal{X} = \{0,1\}^{\ell_{\mathcal{X}}}$.
- *We define* $\mathsf{code}, \mathsf{data}, \mathsf{output}$ *through the following process:*
  1. *Set starting values:*
     (a) $\forall e \in E, \mathsf{data}(e) = \begin{cases} x_i & \text{if } e = e_i \text{ for exactly one } i \in [n] \\ \perp & \text{else} \end{cases}$
     (b) $\forall v \in V, \mathsf{code}(v) = \begin{cases} f_j & \text{if } v = v_j \text{ for exactly one } j \in [m] \\ \perp & \text{else} \end{cases}$
  2. *In topological order of the nodes* $v_j \in \{v_j\}_{j\in[m]}$ *in* G*:*
     (a) $\forall e \in S_{\mathsf{in},j}$*, define* $x_e = \mathsf{data}(e)$*. (If any such* $x_e = \perp$*, skip to the next iteration.)*
     (b) *Compute* $f_j(\{x_e\}_{e\in S_{\mathsf{in},j}}) = \{x_{e'}\}_{e'\in S_{\mathsf{out},j}}$.
     (c) $\forall e' \in S_{\mathsf{out},j}$*, set* $\mathsf{data}(e') = x_{e'}$.

Remark 4.6. For consistency, we will assume that any function which takes $\perp$ (representing an undefined input) as any of its inputs will output $\perp$ for all of its output values. For notational convenience, we may also assume that function output values are ordered such that those with edge type output occur before those with edge type $\mathsf{internal}$.

Now that we have defined the induced graph formed by a valid query set, we will say that a sequence of message and function queries is queried in "topological order" if the order of these queries forms a topological ordering on this graph. (Here, we think of message queries as originating from a source node which outputs the message onto the specified edge.)

Definition 4.7 (Valid Query Set queried in Topological Order)**.** *Consider a sequence of encryption and keygen queries made to an* MHFE *scheme that form a valid query set of the form*

$$W = (\{(x_i, e_i)\}_{i\in[n]}, \{(\mathsf{tid}, f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j\in[m]})$$

with parameters $(\lambda, d_{\mathsf{max}}, \ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \mathsf{type})$*. (For notational convenience, the order that the queries are written as a set* W *above may differ from the order that the queries are actually sequenced.) We say that this sequence of queries was queried in* ***topological order*** *if the following properties are satisfied:*

- *If* $e_i \in S_{\mathsf{in},j}$*, then* (x_i, e_i) *was queried before* $(\mathsf{tid}, f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})$*.*
- *If* $|S_{\mathsf{out},j} \cap S_{\mathsf{in},j'}| > 0$*, then* $(\mathsf{tid}, f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})$ *was queried before* $(\mathsf{tid}, f_{j'}, v_{j'}, S_{\mathsf{in},j'}, S_{\mathsf{out},j'})$*.*

4.1.2 Correctness.

We can now define correctness. We require that the decryption process produces outputs which correspond to the correct distributed computation on the input data.

Definition 4.8 (Correctness of an MHFE scheme)**.** *A public-key* MHFE *scheme* $\mathsf{MHFE} = (\mathsf{MHFE.Setup}, \mathsf{MHFE.EncSetup}, \mathsf{MHFE.Enc}, \mathsf{MHFE.KeyGen}, \mathsf{MHFE.Dec})$ *for* $\mathsf{P/Poly}$ *satisfies* ***correctness*** *if for all polynomials* p*, there exists a negligible function* μ *such that for all* $\lambda \in \mathbb{N}$*, all* $d_{\mathsf{max}}, \ell_{\mathcal{F}}, \ell_{\mathcal{X}}, n, m \leq p(\lambda)$*, all* $\mathsf{tid} \in \{0,1\}^\lambda$*, all functions* $\mathsf{type} : \{0,1\}^\lambda \to \{\mathsf{input}, \mathsf{output}, \mathsf{internal}\}$ *where* $|\mathsf{type}| \leq p(\lambda)$*, and all valid query sets*

$$W = (\{(x_i, e_i)\}_{i\in[n]}, \{(\mathsf{tid}, f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j\in[m]})$$

with parameters $(\lambda, d_{\mathsf{max}}, \ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \mathsf{type})$*, then*

$$\Pr[\forall e \in E^{\mathsf{out}}, \mathsf{data}(e) = \mathsf{data}'(e)] \leq \mu(\lambda)$$

where $E^{\mathsf{out}} = \{e \in E : \mathsf{type}(e) = \mathsf{output}\}$ *and we define* $E, \mathsf{data}, \mathsf{data}'$ *as below:*

- $\pi = (G = (V, E, \mathsf{source}, \mathsf{target}), \mathcal{F}, \mathcal{X}, \mathsf{code}, \mathsf{data})$ *is the induced transcript of* $W = (\{(x_i, e_i)\}_{i\in[n]}, \{(\mathsf{tid}, f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j\in[m]})$*.*
- $\pi' = (G = (V, E, \mathsf{source}, \mathsf{target}), \mathcal{F}', \mathcal{X}', \mathsf{code}', \mathsf{data}')$ *is the induced transcript of* $W' = (\{(\mathsf{ct}_i, e_i)\}_{i\in[n]}, \{(\mathsf{tid}, \mathsf{MHFE.Dec}(\mathsf{sk}_{f_j}, \cdot), v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j\in[m]})$ *where*[8]
  - *values* $\{\mathsf{ct}_i\}_{i\in[n]}$ *and* $\{\mathsf{sk}_j\}_{j\in[m]}$ *are generated using the following process:*
    1. $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{MHFE.Setup}(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$*.*
    2. $\mathsf{ek} \leftarrow \mathsf{MHFE.EncSetup}(\mathsf{mpk})$*.*
    3. $\forall i \in [n], \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{ek}, x_i, e_i)$*.*
    4. $\forall j \in [m]$*,* $\mathsf{sk}_j \leftarrow \mathsf{MHFE.KeyGen}(\mathsf{msk}, \mathsf{tid}, f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})$*.*

[8] We note that since W is a valid query set, then W' is also a valid query set, but with parameters $(\lambda, \ell_{\mathsf{Dec}}, \ell_{\mathsf{ct}}, \mathsf{type})$, where ℓ_{Dec} is the size of $\mathsf{MHFE.Dec}$ and ℓ_{ct} is the size of ciphertexts of MHFE on setup parameters $(1^\lambda, 1^{\ell_{\mathcal{F}}}, \max(1^{\ell_{\mathsf{ct}}}, 1^{\ell_{\mathcal{X}}}), \mathsf{type})$. For W', we allow wires to contain either values in $\{0,1\}^{\ell_{\mathcal{X}}}$ or $\{0,1\}^{\ell_{\mathsf{ct}}}$.

Remark 4.9. While our definition of correctness is stated with respect to a single topology tid, our definition also implies correctness when function keys are simultaneously generated for multiple topologies. In this case, for each topology tid being queried, correctness individually applies to each valid query set $W_{\mathsf{tid}} = (\{(x_i, e_i)\}_{i\in[n]}, \{(\mathsf{tid}, f_{\mathsf{tid},j}, v_{\mathsf{tid},j}, S_{\mathsf{in},\mathsf{tid},j}, S_{\mathsf{out},\mathsf{tid},j})\}_{j\in[m]})$ where the message queries $\{(x_i, e_i)\}_{i\in[n]}$ are shared amongst all the topologies and $\{(\mathsf{tid}, f_{\mathsf{tid},j}, v_{\mathsf{tid},j}, S_{\mathsf{in},\mathsf{tid},j}, S_{\mathsf{out},\mathsf{tid},j})\}_{j\in[m]}$ are the function queries specifically for topology tid. When computing the corresponding ciphertexts and function keys for each modified query set W'_{tid}, $\mathsf{MHFE.Setup}$, $\mathsf{MHFE.EncSetup}$, and encrypting the values $\{(x_i, e_i)\}_{i\in[n]}$ would only need to be done once for all topologies. The master secret key msk and input ciphertexts $\{\mathsf{ct}_i\}_{i\in[n]}$ would be shared amongst all topologies.

4.1.3 Security.

We now define both simulation-based and indistinguishability-based security for MHFE.

SIM Security. Our security is simulation secure and holds for a set of ciphertexts encrypted under a single encryption key ek. We require that for each topology tid, the function and message queries made by the adversary form a valid query set queried in topological order (see Definitions 4.2 and 4.7).

Definition 4.10 (One-Ciphertext, SIM-Security for Public-Key MHFE)**.** *A public-key MHFE scheme* $\mathsf{MHFE} = (\mathsf{MHFE.Setup}, \mathsf{MHFE.EncSetup}, \mathsf{MHFE.Enc}, \mathsf{MHFE.KeyGen}, \mathsf{MHFE.Dec})$ *for* $\mathsf{P/Poly}$ *is one-ciphertext, SIM-secure if there exists a negligible function* μ *and a PPT simulator* $\mathsf{Sim} = (\mathsf{Sim.Setup}, \mathsf{Sim.Enc}, \mathsf{Sim.KeyGen})$[9] *such that for all* $\lambda \in \mathbb{N}$ *and all PPT adversaries* $\mathcal{A}$*,*

$$\left|\Pr[\mathsf{Real}^{\mathsf{MHFE}}_{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathsf{Ideal}^{\mathsf{MHFE}}_{\mathcal{A}}(1^\lambda) = 1]\right| \leq \mu(\lambda)$$

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

$\mathsf{Real}^{\mathsf{MHFE}}_{\mathcal{A}}(1^\lambda)$

1. ***Parameters:*** *$\mathcal{A}$ takes as input 1^λ, and outputs parameters $(1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.*
2. ***Setup:***
   (a) $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{MHFE.Setup}(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.
   (b) *Send* mpk *to* $\mathcal{A}$.
3. ***Encryption Setup:*** $\mathsf{ek} \leftarrow \mathsf{MHFE.EncSetup}(\mathsf{mpk})$.
4. *For a polynomial number of rounds, the adversary can do either one of the following in each round:*
   (a) ***Function Query:***
      i. *$\mathcal{A}$ outputs function query* $(\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$.
      ii. $\mathsf{sk}_{\mathsf{tid},v} \leftarrow \mathsf{MHFE.KeyGen}(\mathsf{msk}, \mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$.
      iii. *Send* $\mathsf{sk}_{\mathsf{tid},v}$ *to* $\mathcal{A}$.
   (b) ***Challenge Message Query:***
      i. *$\mathcal{A}$ outputs challenge message query* (x, e).

[9] We assume that Sim is stateful and shares state between its various algorithms.

ii. $\mathsf{ct}_e \leftarrow \mathsf{MHFE.Enc}(\mathsf{ek}, x, e)$.

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

5. ***Adversary's Output****:* $\mathcal{A}$ *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 for any tid *queried by the adversary,*

$$W = (\{(x_i, e_i)\}_{i\in[n]}, \{(\mathsf{tid}, f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j\in[m]})$$

is ever ***not*** *a valid query set with parameters* $(\lambda, d_{\mathsf{max}}, \ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \mathsf{type})$ *queried in* ***topological order****, where* $\{(x_i, e_i)\}_{i\in[n]}$ *are all challenge message queries made by* $\mathcal{A}$ *and the set* $\{(\mathsf{tid}, f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j\in[m]}$ *are the function queries made by* $\mathcal{A}$ *to topology id* tid.

$\mathsf{Ideal}^{\mathsf{MHFE}}_{\mathcal{A}}(1^\lambda)$

1. ***Parameters****:* $\mathcal{A}$ *takes as input* 1^λ*, and outputs parameters* $(1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.

2. ***Setup****:*

   (a) $\mathsf{mpk} \leftarrow \mathsf{Sim.Setup}(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.

   (b) Send mpk *to* $\mathcal{A}$.

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

   (a) ***Function Query****:*

      i. $\mathcal{A}$ *outputs function query* $(\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$.

      ii. Compute output:

         A. For $e \in S_{\mathsf{in}}$*, if* $x_{\mathsf{tid},e}$ *is not already defined,*

            - *If* (x, e) *has been queried as a challenge message query, set* $x_{\mathsf{tid},e} = x$.
            - *Else, set* $x_{\mathsf{tid},e} = \perp$.[a]

         B. $(\{x_\alpha\}_{\alpha\in S^{\mathsf{out}}_{\mathsf{out}}}, \{x_\beta\}_{\beta\in S^{\mathsf{int}}_{\mathsf{out}}}) = f(\{x_e\}_{e\in S_{\mathsf{in}}})$.

      iii. $\mathsf{sk}_{\mathsf{tid},v} \leftarrow \mathsf{Sim.KeyGen}(\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, \{x_\alpha\}_{\alpha\in S^{\mathsf{out}}_{\mathsf{out}}}.)$.

      iv. Send $\mathsf{sk}_{\mathsf{tid},v}$ *to* $\mathcal{A}$.

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

      i. $\mathcal{A}$ *outputs challenge message query* (x, e).

      ii. $\mathsf{ct}_e \leftarrow \mathsf{Sim.Enc}(e)$.

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

4. ***Adversary's Output****:* $\mathcal{A}$ *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 for any tid *queried by the adversary,*

$$W = (\{(x_i, e_i)\}_{i\in[n]}, \{(\mathsf{tid}, f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j\in[m]})$$

is ever ***not*** *a valid query set with parameters* $(\lambda, d_{\mathsf{max}}, \ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \mathsf{type})$ *queried in* ***topological order****, where* $\{(x_i, e_i)\}_{i\in[n]}$ *are all challenge message queries made by* $\mathcal{A}$ *and the set* $\{(\mathsf{tid}, f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j\in[m]}$ *are the function queries made by* $\mathcal{A}$ *to topology id* tid*.*

[a]Topological order only constrains the relative order of queries that reference a common edge, so an edge in S_{in} does not need to have any producing query at all; rather than aborting in this case, we rely on the $\perp$-propagation convention (Remark 4.6) to keep the real and ideal experiments consistent, intentionally allowing a hop key to be requested before, or without, all of its input data ever arriving.

IND Security. We also describe an indistinguishability-based notion of security in which two sets of encrypted messages are indistinguishable if they produce the same output wire values. In this definition, we also allow for multiple sets of challenge message queries to be encrypted under different encryption keys ek. To indicate which encryption key each message query should be encrypted under, the adversary will also include a message id cid with each of its message queries.

Remark 4.11. Our construction also achieves the definition of IND-security given below. This is because a one-ciphertext, SIM-secure public-key MHFE scheme immediately implies a one-ciphertext, IND-secure public-key MHFE scheme, which implies a (multiple-ciphertext) IND-Secure public-key MHFE scheme via a standard hybrid argument.

Definition 4.12 (IND-Security for Public-Key MHFE)**.** *A public-key MHFE scheme* MHFE *for* P/Poly *is IND-secure if there exists a negligible function* μ *such that for all* $\lambda \in \mathbb{N}$ *and all PPT adversaries* $\mathcal{A}$*,*

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

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

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

1. ***Parameters****:* $\mathcal{A}$ *takes as input* 1^λ*, and outputs parameters* $(1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$*.*
2. ***Setup****:*
   (a) $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{MHFE.Setup}(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$*.*
   (b) *Send* mpk *to* $\mathcal{A}$*.*
3. *For a polynomial number of rounds, the adversary can do either one of the following in each round:*
   (a) ***Function Query****:*
      i. $\mathcal{A}$ *outputs function query* $(\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$*.*
      ii. $\mathsf{sk}_{\mathsf{tid},v} \leftarrow \mathsf{MHFE.KeyGen}(\mathsf{msk}, \mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$*.*
      iii. Send $\mathsf{sk}_{\mathsf{tid},v}$ *to* $\mathcal{A}$*.*
   (b) ***Challenge Message Query****:*
      i. $\mathcal{A}$ *outputs challenge message query* $(\mathsf{cid}, x_0, x_1, e)$*.*
      ii. If this is the first message query made to message identity cid*,*
         A. $\mathsf{ek}_{\mathsf{cid}} \leftarrow \mathsf{MHFE.EncSetup}(1^\lambda)$*.*
      iii. $\mathsf{ct}_{\mathsf{cid},e,b} \leftarrow \mathsf{MHFE.Enc}(\mathsf{ek}_{\mathsf{cid}}, x_b, e)$*.*

iv. Send $\mathsf{ct}_{\mathsf{cid},e,b}$ *to* $\mathcal{A}$.

4. ***Adversary's Output****:* $\mathcal{A}$ *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 for any cid *or* tid *queried by the adversary and for any* $b \in \{0,1\}$

$$W_{\mathsf{tid},\mathsf{cid},b} = (\{(\mathsf{cid}, x_{i,b}, e_i)\}_{i\in[n]}, \{(\mathsf{tid}, f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j\in[m]})$$

is ever ***not*** *a valid query set with parameters* $(\lambda, d_{\mathsf{max}}, \ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \mathsf{type})$ *queried in* ***topological order****, where* $\{(\mathsf{cid}, x_{i,0}, x_{i,1}, e_i)\}_{i\in[n]}$ *are the challenge message queries by* $\mathcal{A}$ *to message identity* cid*, and* $\{(\mathsf{tid}, f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j\in[m]}$ *are the function queries made by* $\mathcal{A}$ *to topology id* tid*.*

We also immediately halt and output 0 *if for any* cid *or* tid *queried by the adversary, we produce different output wire values when encrypting in the* $b = 0$ *case as opposed to the* $b = 1$ *case. That is, we immediately halt and output* 0*, if it is ever the case that for any* cid *or* tid *queried by the adversary*

$$\exists e \in E^{\mathsf{out}}_{\mathsf{tid}} \text{ such that } \mathsf{data}_{\mathsf{tid},\mathsf{cid},0}(e) \neq \mathsf{data}_{\mathsf{tid},\mathsf{cid},1}(e)$$

where $E^{\mathsf{out}}_{\mathsf{tid}} = \{e \in E_{\mathsf{tid}} : \mathsf{type}(e) = \mathsf{output}\}$ *and where* $E_{\mathsf{tid}}, \mathsf{data}_{\mathsf{tid},\mathsf{cid},b}$ *are taken from the induced transcript* $\pi_{\mathsf{tid},\mathsf{cid},b} = (G_{\mathsf{tid}} = (V_{\mathsf{tid}}, E_{\mathsf{tid}}, \mathsf{source}_{\mathsf{tid}}, \mathsf{target}_{\mathsf{tid}}), \mathcal{F}, \mathcal{X}, \mathsf{code}_{\mathsf{tid}}, \mathsf{data}_{\mathsf{tid},\mathsf{cid},b})$ *of the valid query set* $W_{\mathsf{tid},\mathsf{cid},b}$ *defined above.*

4.2 Multi-Hop Garbled Circuits (MHGC)

Intuitively, a multi-hop garbled circuit is a garbled circuit that can be constructed in an incremental fashion. It can be interpreted as a secret-key version of MHFE that only allows for a single topology, or, alternatively, as an adaptation of reactive garbling [NR16]. We define it below:

Definition 4.13 (Multi-Hop Garbled Circuit)**.** *A Multi-Hop Garbled Circuit scheme (MHGC) for* P/Poly *is a tuple of PPT algorithms consisting of* MHGC = (MHGC.Setup, MHGC.GarbInp, MHGC.GarbCirc, MHGC.Eval) *defined as follows:*

- MHGC.Setup$(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$*: takes as input the security parameter* λ*, a maximum degree* d_{max}*, a function description size* $\ell_{\mathcal{F}}$*, a message size* $\ell_{\mathcal{X}}$*, and a function* $\mathsf{type} : \mathcal{E} \to \{\mathsf{input}, \mathsf{output}, \mathsf{internal}\}$ *mapping edges to edge types, where*
  - $\mathcal{F}$ *is the function space containing all circuits with descriptions in* $\{0,1\}^{\ell_{\mathcal{F}}}$*,*
  - $\mathcal{X} = \{0,1\}^{\ell_{\mathcal{X}}}$ *is the message space,*
  - $\mathcal{V} = \{0,1\}^\lambda$ *is the vertex space,*
  - $\mathcal{E} = \{0,1\}^\lambda$ *is the edge space, with* $\mathcal{E}^{\mathsf{inp}}, \mathcal{E}^{\mathsf{out}}, \mathcal{E}^{\mathsf{int}}$ *being the restrictions of* $\mathcal{E}$ *to edges with type* input, output, internal *respectively.*

  and outputs the master secret key msk*.*
- MHGC.GarbInp(msk, x, e)*: takes as input the master secret key* msk*, a garbled input* $x \in \mathcal{X}$*, and an edge* $e \in \mathcal{E}^{\mathsf{inp}}$*, and outputs a message encoding* $\widehat{X}_e$ *of* x*.*
- MHGC.GarbCirc$(\mathsf{msk}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$*: takes as input the master secret key* msk*, a function* $f \in \mathcal{F}$ *with* $f : \mathcal{X}^{|S_{\mathsf{in}}|} \to \mathcal{X}^{|S_{\mathsf{out}}|}$*, a vertex label* $v \in \mathcal{V}$*, and an ordered set of incoming edges* $S_{\mathsf{in}} \subseteq \mathcal{E}^{\mathsf{inp}} \cup \mathcal{E}^{\mathsf{int}}$ *and outgoing edges* $S_{\mathsf{out}} \subseteq \mathcal{E}^{\mathsf{out}} \cup \mathcal{E}^{\mathsf{int}}$*, and outputs a garbled circuit* $\widehat{F}_{\mathsf{tid},v,S_{\mathsf{in}},S_{\mathsf{out}}}$*.*

- $\mathsf{MHGC.Dec}(\widehat{F}_{\mathsf{tid},v,S_{\mathsf{in}},S_{\mathsf{out}}}, \{\widehat{X}_e\}_{e\in S_{\mathsf{in}}})$: *takes as input a garbled circuit* $\widehat{F}_{\mathsf{tid},v,S_{\mathsf{in}},S_{\mathsf{out}}}$ *and garbled inputs* $\{\widehat{X}_e\}_{e\in S_{\mathsf{in}}}$ *corresponding to the incoming edges to vertex* v*, and outputs values* $(\{x_\alpha\}_{\alpha\in S^{\mathsf{out}}_{\mathsf{out}}}, \{\widehat{X}_\beta\}_{\beta\in S^{\mathsf{int}}_{\mathsf{out}}})$ *corresponding to the outgoing edges of vertex* v*, where* $S^{\mathsf{out}}_{\mathsf{out}}$ *and* $S^{\mathsf{int}}_{\mathsf{out}}$ *are the restrictions of* S_{out} *to edges of type* output *and* internal *respectively.*

MHGC schemes must satisfy correctness (Definition 4.15) and security (Definition 4.16).

Remark 4.14. We define valid query sets, induced transcripts, and topological orderings on valid query sets for MHGC in the same way as the corresponding notions for MHFE (Definitions 4.2, 4.5, and 4.7) except that

- We remove the topology id tid since we only have a single topology.
- "Message queries" refer to calls to MHGC.GarbInp rather than calls to MHFE.Enc.
- "Function queries" refer to calls to MHGC.GarbCirc rather than calls to MHFE.KeyGen.

We require that the garbled evaluation process produces outputs which correspond to the correct distributed computation on the input data.

Definition 4.15 (Correctness of a MHGC scheme)**.** *A* MHGC *scheme* $\mathsf{MHGC} = (\mathsf{MHGC.Setup}, \mathsf{MHGC.GarbInp}, \mathsf{MHGC.GarbCirc}, \mathsf{MHGC.Eval})$ *for* P/Poly *satisfies* ***correctness*** *if for all polynomials* p*, there exists a negligible function* μ *such that for all* $\lambda \in \mathbb{N}$*, all* $d_{\mathsf{max}}, \ell_{\mathcal{F}}, \ell_{\mathcal{X}}, n, m \le p(\lambda)$*, all functions* $\mathsf{type} : \{0,1\}^\lambda \to \{\mathsf{input}, \mathsf{output}, \mathsf{internal}\}$ *where* $|\mathsf{type}| \le p(\lambda)$*, and all valid query sets*

$$W = (\{(x_i, e_i)\}_{i\in[n]}, \{(f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j\in[m]})$$

with parameters $(\lambda, d_{\mathsf{max}}, \ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \mathsf{type})$*, then*

$$\Pr[\forall e \in E^{\mathsf{out}}, \mathsf{data}(e) = \mathsf{data}'(e)] \le \mu(\lambda)$$

where $E^{\mathsf{out}} = \{e \in E : \mathsf{type}(e) = \mathsf{output}\}$ *and we define* $E, \mathsf{data}, \mathsf{data}'$ *as below:*

- $\pi = (G = (V, E, \mathsf{source}, \mathsf{target}), \mathcal{F}, \mathcal{X}, \mathsf{code}, \mathsf{data})$ *is the induced transcript of* $W = (\{(x_i, e_i)\}_{i\in[n]}, \{(f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j\in[m]})$.
- $\pi' = (G = (V, E, \mathsf{source}, \mathsf{target}), \mathcal{F}', \mathcal{X}', \mathsf{code}', \mathsf{data}')$ *is the induced transcript of* $W' = (\{(\widehat{X}_i, e_i)\}_{i\in[n]}, \{(\mathsf{tid}, \mathsf{MHGC.Eval}(\widehat{F}_j, \cdot), v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j\in[m]})$ *where*[10]
  - *values* $\{\widehat{X}_i\}_{i\in[n]}$ *and* $\{\widehat{F}_j\}_{j\in[m]}$ *are generated using the following process:*
    1. $\mathsf{msk} \leftarrow \mathsf{MHGC.Setup}(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.
    2. $\forall i \in [n], \widehat{X}_i \leftarrow \mathsf{MHGC.GarbInp}(\mathsf{msk}, x_i, e_i)$.
    3. $\forall j \in [m], \widehat{F}_j \leftarrow \mathsf{MHGC.KeyGen}(\mathsf{msk}, f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})$.

Security requires that the entire process should be simulatable without knowing any of the unrevealed garbled wire values.

[10] We note that since W is a valid query set, then W' is also a valid query set, but with parameters $(\lambda, \ell_{\mathsf{Eval}}, \max(\ell_{\widehat{\mathcal{X}}}, \ell_{\mathcal{X}}), \mathsf{type})$, where ℓ_{Eval} is the size of MHGC.Eval and $\ell_{\widehat{\mathcal{X}}}$ is the size of garbled inputs of MHGC on setup parameters $(1^\lambda, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$. For W', we allow wires to contain either values in $\{0,1\}^{\ell_{\mathcal{X}}}$ or $\{0,1\}^{\widehat{\mathcal{X}}}$.

Definition 4.16 (SIM-Security for MHGC)**.** *A* MHGC *scheme* $\mathsf{MHGC} = (\mathsf{MHGC.Setup}, \mathsf{MHGC.GarbInp}, \mathsf{MHGC.GarbCirc}, \mathsf{MHGC.Eval})$ *for* P/Poly *is SIM-secure if there exists a negligible function* μ *and a PPT simulator* $\mathsf{Sim} = (\mathsf{Sim.Setup}, \mathsf{Sim.GarbInp}, \mathsf{Sim.GarbCirc})$[11] *such that for all* $\lambda \in \mathbb{N}$ *and all PPT adversaries* $\mathcal{A}$*,*

$$\left|\Pr[\mathsf{Real}_{\mathcal{A}}^{\mathsf{MHGC}}(1^\lambda) = 1] - \Pr[\mathsf{Ideal}_{\mathcal{A}}^{\mathsf{MHGC}}(1^\lambda) = 1]\right| \leq \mu(\lambda)$$

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

$\mathsf{Real}_{\mathcal{A}}^{\mathsf{MHGC}}(1^\lambda)$

1. ***Parameters****:* $\mathcal{A}$ *takes as input* 1^λ*, and outputs parameters* $(1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$*.*
2. ***Setup****:* $(\mathsf{msk}) \leftarrow \mathsf{MHGC.Setup}(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$*.*
3. *For a polynomial number of rounds, the adversary can do either one of the following in each round:*
   (a) ***Function Query****:*
      i. $\mathcal{A}$ *outputs function query* $(f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$*.*
      ii. $\widehat{F}_v \leftarrow \mathsf{MHGC.GarbCirc}(\mathsf{msk}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$*.*
      iii. *Send* $\widehat{F}_v$ *to* $\mathcal{A}$*.*
   (b) ***Message Query****:*
      i. $\mathcal{A}$ *outputs message query* (x, e)*.*
      ii. $\widehat{X}_e \leftarrow \mathsf{MHGC.GarbInp}(\mathsf{msk}, x, e)$*.*
      iii. *Send* $\widehat{X}_e$ *to* $\mathcal{A}$*.*
4. ***Adversary's Output****:* $\mathcal{A}$ *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

$$W = (\{(x_i, e_i)\}_{i\in[n]}, \{(f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j\in[m]})$$

is ever ***not*** *a valid query set with parameters* $(\lambda, d_{\mathsf{max}}, \ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \mathsf{type})$ *queried in* ***topological order****, where* $\{(x_i, e_i)\}_{i\in[n]}$ *are the message queries made by* $\mathcal{A}$ *and* $\{(f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j\in[m]}$ *are the function queries made by* $\mathcal{A}$*.*

$\mathsf{Ideal}_{\mathcal{A}}^{\mathsf{MHGC}}(1^\lambda)$

1. ***Parameters****:* $\mathcal{A}$ *takes as input* 1^λ*, and outputs parameters* $(1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$*.*
2. ***Setup****: We run* $\mathsf{Sim.Setup}(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$*.*
3. *For a polynomial number of rounds, the adversary can do either one of the following in each round:*
   (a) ***Function Query****:*
      i. $\mathcal{A}$ *outputs function query* $(f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$*.*

[11] We assume that Sim is stateful and shares state between its various algorithms.

ii. Compute output:

A. For $e \in S_{\mathsf{in}}$, if x_e is not already defined,

- *If (x, e) has been queried as a challenge message query, set $x_e = x$.*
- *Else, set $x_e = \bot$.*

B. $(\{x_\alpha\}_{\alpha \in S^{\mathsf{out}}_{\mathsf{out}}}, \{x_\beta\}_{\beta \in S^{\mathsf{int}}_{\mathsf{out}}}) = f(\{x_e\}_{e \in S_{\mathsf{in}}})$.

iii. $\widehat{F}_v \leftarrow \mathsf{Sim.GarbCirc}(f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, \{x_\alpha\}_{\alpha \in S^{\mathsf{out}}_{\mathsf{out}}})$.

iv. Send $\widehat{F}_v$ to $\mathcal{A}$.

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

i. $\mathcal{A}$ outputs message query (x, e).

ii. $\widehat{X}_e \leftarrow \mathsf{Sim.GarbInp}(e)$.

iii. Send $\widehat{X}_e$ to $\mathcal{A}$.

4. ***Adversary's Output****: $\mathcal{A}$ 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

$$W = (\{(x_i, e_i)\}_{i \in [n]}, \{(f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j \in [m]})$$

is ever ***not*** *a valid query set with parameters $(\lambda, d_{\mathsf{max}}, \ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \mathsf{type})$ queried in* ***topological order****, where $\{(x_i, e_i)\}_{i \in [n]}$ are the message queries made by $\mathcal{A}$ and $\{(f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j \in [m]}$ are the function queries made by $\mathcal{A}$.*

5 Constructing a Multi-Hop Garbled Circuit

In this section, we construct a multi-hop garbled circuit scheme (MHGC).

Theorem 5.1. *Assuming* OWF*s, there exists a multi-hop garbled circuit scheme* MHGC*.*

To prove Theorem 5.1, we build a MHGC scheme from the following tools, which can each be built from OWFs using standard techniques[Gol01, Gol09, Yao86].

Tools.

- $\mathsf{PRF} = (\mathsf{PRF.Setup}, \mathsf{PRF.Eval})$: A secure pseudorandom function family.
- $\mathsf{PRF}_p = (\mathsf{PRF}_p\mathsf{.Setup}, \mathsf{PRF}_p\mathsf{.Eval})$: A secure pseudorandom function family.
- $\mathsf{SKE} = (\mathsf{SKE.Setup}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$: A secure secret-key encryption scheme.
- $\mathsf{GC} = (\mathsf{GC.Garble}, \mathsf{GC.Eval})$: A secure garbling scheme.

5.1 Parameters

On security parameter λ, maximum degree d_{max}, function size $\ell_{\mathcal{F}}$, message size $\ell_{\mathcal{X}}$, and type function type, we instantiate our primitives with the following parameters:

- We instantiate PRF with security parameter λ, input size $\lambda + \log(\ell_{\mathcal{X}}) + 1$, and output size λ.
- We instantiate PRF_p with security parameter λ, input size λ, and output size $\ell_{\mathcal{X}}$.

- SKE: We instantiate SKE with security parameter λ, which we will use to encrypt messages of length λ.
- GC: We instantiate GC with security parameter λ. We will use GC to garble circuits of the form $P[f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, \{p_e\}_{e \in S_{\mathsf{in}}}, \{p_\beta, \mathsf{sk}_{\beta,k,b}\}_{\beta \in S^{\mathsf{int}}_{\mathsf{out}}, k \in [\ell_{\mathcal{X}}], b \in \{0,1\}}]$ as defined in the construction below, where $S_{\mathsf{in}}, S_{\mathsf{out}} \subseteq \{0,1\}^\lambda$, $|S_{\mathsf{in}}|, |S_{\mathsf{out}}| \leq d_{\mathsf{max}}$, $f \in \{0,1\}^{\ell_{\mathcal{F}}}$, each $p_e, p_\beta, \widetilde{x}_e \in \{0,1\}^{\ell_{\mathcal{X}}}$, and each $\mathsf{sk}_{\beta,k,b}$ is a secret key of SKE as defined above. We let ℓ_P denote the size of such circuits. Recall that each of the input labels of a garbled circuit are size λ.

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

5.2 Construction

We now construct our multi-hop garbled circuit MHGC.

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

Notation. Recall that if S is any set of edges and we have an edge labeling function $\mathsf{type} : S \to \{\mathsf{input}, \mathsf{output}, \mathsf{internal}\}$, then we define $S^{\mathsf{inp}}, S^{\mathsf{out}}, S^{\mathsf{int}}$ to be the restrictions of S to edges with type $\mathsf{input}, \mathsf{output}, \mathsf{internal}$ respectively, i.e. $S^{\mathsf{inp}} := \{e \in S : \mathsf{type}(e) = \mathsf{input}\}$, $S^{\mathsf{out}} := \{e \in S : \mathsf{type}(e) = \mathsf{output}\}$, $S^{\mathsf{int}} := \{e \in S : \mathsf{type}(e) = \mathsf{internal}\}$.

MHGC**:**

- $\mathsf{MHGC.Setup}(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$:
  1. $K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
  2. $K_p \leftarrow \mathsf{PRF}_p\mathsf{.Setup}(1^\lambda)$.
  * Throughout, for $e \in \{0,1\}^\lambda$, $k \in [\ell_{\mathcal{X}}]$, $b \in \{0,1\}$, we will define
  $$\mathsf{sk}_{e,k,b} = \mathsf{SKE.Setup}(1^\lambda, \mathsf{PRF.Eval}(K, (e,k,b)))$$
  $$p_e = \mathsf{PRF}_p\mathsf{.Eval}(K_p, e)$$
  3. Output $\mathsf{MSK} = (K, K_p, \mathsf{type})$.
- $\mathsf{MHGC.GarbInp}(\mathsf{MSK}, x_e, e)$:
  1. Compute $(p_e, \{\mathsf{sk}_{e,k,b}\}_{k\in[\ell_{\mathcal{X}}],b\in\{0,1\}})$ from $\mathsf{MSK} = (K, K_p)$.
  2. $\widetilde{x}_e = x_e \oplus p_e$.
  3. Output $\widehat{X}_e = (\widetilde{x}_e, \{\mathsf{sk}_{e,k,\widetilde{x}_e[k]}\}_{k\in[\ell_{\mathcal{X}}]})$.
- $\mathsf{MHGC.GarbCirc}(\mathsf{MSK}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$:
  1. **Compute Keys:**
     (a) For $e \in S_{\mathsf{in}}$, compute $(p_e, \{\mathsf{sk}_{e,k,b}\}_{k\in[\ell_{\mathcal{X}}],b\in\{0,1\}})$ from $\mathsf{MSK} = (K, K_p)$.
     (b) For $\beta \in S^{\mathsf{int}}_{\mathsf{out}}$, compute $(p_\beta, \{\mathsf{sk}_{\beta,k,b}\}_{k\in[\ell_{\mathcal{X}}],b\in\{0,1\}})$ from $\mathsf{MSK} = (K, K_p)$.
  2. **Garble Circuit**:
     (a) Let $P_{f,v} = P[f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, \{p_e\}_{e\in S_{\mathsf{in}}}, \{p_\beta, \mathsf{sk}_{\beta,k,b}\}_{\beta\in S^{\mathsf{int}}_{\mathsf{out}},k\in[\ell_{\mathcal{X}}],b\in\{0,1\}}]$ as defined in Figure 4.

$P[f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, \{p_e\}_{e\in S_{\mathsf{in}}}, \{p_\beta, \mathsf{sk}_{\beta,k,b}\}_{\beta\in S^{\mathsf{int}}_{\mathsf{out}},k\in[\ell_{\mathcal{X}}],b\in\{0,1\}}](\{\widetilde{x}_e\}_{e\in S_{\mathsf{in}}})$:

i. For $e \in S_{\mathsf{in}}$, $x_e = \widetilde{x}_e \oplus p_e$.

ii. $(\{x_\alpha\}_{\alpha\in S^{\mathsf{out}}_{\mathsf{out}}}, \{x_\beta\}_{\beta\in S^{\mathsf{int}}_{\mathsf{out}}}) = f(\{x_e\}_{e\in S_{\mathsf{in}}})$.

iii. For $\beta \in S^{\mathsf{int}}_{\mathsf{out}}$,
   – $\widetilde{x}_\beta = x_\beta \oplus p_\beta$.
   – $\widehat{X}_\beta = (\widetilde{x}_\beta, \{\mathsf{sk}_{\beta,k,\widetilde{x}_\beta[k]}\}_{k\in[\ell_{\mathcal{X}}]})$.

iv. Output $(\{x_\alpha\}_{\alpha\in S^{\mathsf{out}}_{\mathsf{out}}}, \{\widehat{X}_\beta\}_{\beta\in S^{\mathsf{int}}_{\mathsf{out}}})$.

Figure 4: Definition of $P_{f,v}$

(b) $(\widetilde{P}_{f,v}, \{\mathsf{lab}_{e,k,b}\}_{e\in S_{\mathsf{in}},k\in[\ell_{\mathcal{X}}],b\in\{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, P_{f,v})$

3. **Encrypt Labels**:
   (a) For $e \in S_{\mathsf{in}}, k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$, compute $\mathsf{ct}_{e,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{e,k,b}, \mathsf{lab}_{e,k,b})$.
4. Output $\widehat{F}_{v,S_{\mathsf{in}},S_{\mathsf{out}}} = (\widetilde{P}_{f,v}, \{\mathsf{ct}_{e,k,b}\}_{e\in S_{\mathsf{in}},k\in[\ell_{\mathcal{X}}],b\in\{0,1\}})$.

- $\mathsf{MHGC.Eval}(\widehat{F}_{v,S_{\mathsf{in}},S_{\mathsf{out}}}, \{\widehat{X}_e\}_{e\in S_{\mathsf{in}}})$:

1. Parse $\widehat{F}_{v,S_{\mathsf{in}},S_{\mathsf{out}}} = (\widetilde{P}_{f,v}, \{\mathsf{ct}_{e,k,b}\}_{e\in S_{\mathsf{in}},k\in[\ell_{\mathcal{X}}],b\in\{0,1\}})$
   and $\{\widehat{X}_e\}_{e\in S_{\mathsf{in}}} = \{(\widetilde{x}_e, \{\mathsf{sk}_{e,k,\widetilde{x}_e[k]}\}_{k\in[\ell_{\mathcal{X}}]})\}_{e\in S_{\mathsf{in}}}$.
2. **Decrypt Labels**:
   (a) For $e \in S_{\mathsf{in}}$, $k \in [\ell_{\mathcal{X}}]$, compute $\mathsf{lab}_{e,k,\widetilde{x}_e[k]} = \mathsf{SKE.Dec}(\mathsf{sk}_{e,k,\widetilde{x}_e[k]}, \mathsf{ct}_{e,k,\widetilde{x}_e[k]})$.
3. **Evaluate Garbled Circuit**:
   (a) $(\{x_\alpha\}_{\alpha\in S^{\mathsf{out}}_{\mathsf{out}}}, \{\widehat{X}_\beta\}_{\beta\in S^{\mathsf{int}}_{\mathsf{out}}}) = \mathsf{GC.Eval}(\widetilde{P}_{f,v}, \{\mathsf{lab}_{e,k,\widetilde{x}_e[k]}\}_{e\in S_{\mathsf{in}},k\in[\ell_{\mathcal{X}}]})$.
4. Output $(\{x_\alpha\}_{\alpha\in S^{\mathsf{out}}_{\mathsf{out}}}, \{\widehat{X}_\beta\}_{\beta\in S^{\mathsf{int}}_{\mathsf{out}}})$.

5.3 Correctness

Efficiency: Using our discussion above on parameters, it is easy to see that the size and runtime of all algorithms of our MHGC scheme on security parameter 1^λ, maximum degree d_{max}, function size $\ell_{\mathcal{F}}$, message size $\ell_{\mathcal{X}}$, and type function type are $\mathsf{poly}(\lambda, d_{\mathsf{max}}, \ell_{\mathcal{F}}, \ell_{\mathcal{X}}, |\mathsf{type}|)$.

Correctness: We can prove correctness by considering a topological ordering on the nodes of the induced graph formed by the message and function queries, and proving inductively that $\mathsf{GC.Eval}$ outputs the correct values at each node.

Thus, suppose that at some node, all upstream values have been correctly computed. Then, when we compute $\mathsf{GC.Eval}$ on this node, by the correctness of the SKE scheme, we will correctly recover the labels $\{\mathsf{lab}_{e,k,\widetilde{x}_e[k]}\}_{e\in S_{\mathsf{in}}}$ corresponding to input value $\{\widetilde{x}_e\}_{e\in S_{\mathsf{in}}}$, where $\widetilde{x}_e = x_e \oplus p_e$ and where x_e is the correct value for wire e. By correctness of GC, running $\mathsf{GC.Eval}(\widetilde{P}_{f,v}, \{\mathsf{lab}_{e,k,\widetilde{x}_e[k]}\}_{e\in S_{\mathsf{in}},k\in[\ell_{\mathcal{X}}]})$ will produce $P_{f,v}(\{\widetilde{x}_e\}_{e\in S_{\mathsf{in}}}) = (\{x_\alpha\}_{\alpha\in S^{\mathsf{out}}_{\mathsf{out}}}, \{\widehat{X}_\beta\}_{\beta\in S^{\mathsf{int}}_{\mathsf{out}}})$ where each x_α is the correct output value for wire α and where each $\widehat{X}_\beta$ contains the corresponding secret keys for $\widetilde{x}_\beta = x_\beta \oplus p_\beta$, allowing the process to continue.

5.4 Security

In this section, we prove that MHGC is a secure multi-hop garbled circuit scheme.

Remark 5.3. For all hybrids, we will assume that when running the hybrid, we immediately halt and output 0 if the adversary ever aborts or if

$$W = (\{(x_i, e_i)\}_{i\in[n]}, \{(f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j\in[m]})$$

is ever **not** a valid query set with parameters $(\lambda, d_{\mathsf{max}}, \ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \mathsf{type})$ queried in **topological order**, where $\{(x_i, e_i)\}_{i\in[n]}$ are the message queries made by $\mathcal{A}$ and $\{(f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j\in[m]}$ are the function queries made by $\mathcal{A}$.

Remark 5.4. For notational convenience, we will assume that the vertex labels v are queried in sequential order, starting from vertex $v = 1$ and incrementing upward. Note that this can easily be accomplished via renaming.

5.4.1 Proof Overview.

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

- $\mathbf{Hybrid}_0^{\mathcal{A}}(\lambda)$: This is the real world experiment where we use the algorithms defined in our construction.
- $\mathbf{Hybrid}_1^{\mathcal{A}}(\lambda)$: We compute the SKE secret keys $\mathsf{sk}_{e,k,b}$ and one-time-pads p_e using uniform randomness, rather than PRFs. Indistinguishability follows from the security of PRF and PRF_p.
- We now go through the following hybrids for $t \in [\mathsf{Bound}_{\mathcal{A}}]$ where $\mathsf{Bound}_{\mathcal{A}}$ is a bound on the runtime of $\mathcal{A}$, and thus an implicit bound on the number of message and function queries made by the adversary. On iteration t, the goal is to switch to a hybrid where we simulate the tth usage of the garbled circuit scheme GC.
  - $\mathbf{Hybrid}_{2,t,0}^{\mathcal{A}}(\lambda)$: For vertices $v < t$, we generate the garbled circuit $\widetilde{P}_{f,v}$ and the input labels corresponding to the padded incoming edge values $\{\widetilde{x}_e\}_{e \in S_{\mathsf{in}}}$ using the simulator for the garbling scheme. Since we are simulating the garbled circuit for $P_{f,v}$, we no longer need to embed the SKE secret keys $\{\mathsf{sk}_{\beta,k,1-\widetilde{x}_\beta[k]}\}_{\beta \in S_{\mathsf{out}}^{\mathsf{int}}, k \in [\ell_{\mathcal{X}}]}$ for values **not** corresponding to the outgoing internal edge values $\{\widetilde{x}_\beta\}_{\beta \in S_{\mathsf{out}}^{\mathsf{int}}}$ into $P_{f,v}$. When $t = 1$, this hybrid is identical to $\mathbf{Hybrid}_1^{\mathcal{A}}$.
  - $\mathbf{Hybrid}_{2,t,1}^{\mathcal{A}}(1^\lambda)$: For vertex v, we change the encryptions of input labels $\{\mathsf{lab}_{e,k,1-\widetilde{x}_e}[k]\}_{e \in S_{\mathsf{in}}, k \in [\ell_{\mathcal{X}}]}$ which do not correspond to the input values $\{\widetilde{x}_e\}_{e \in S_{\mathsf{in}}}$ to encryptions of $\perp$. This follows from the security of the SKE scheme since we removed the usage of the corresponding SKE keys in the previous hybrid.
  - $\mathbf{Hybrid}_{2,t,2}^{\mathcal{A}}(\lambda)$: We simulate the garbled circuit for $v = t$. This is feasible because we have just removed the unused input labels from the adversary's view. Indistinguishability follows from the security of the garbling scheme. Additionally, $\mathbf{Hybrid}_{2,t,2}^{\mathcal{A}}$ and $\mathbf{Hybrid}_{2,t+1,0}^{\mathcal{A}}$ are identical.
- $\mathbf{Hybrid}_3^{\mathcal{A}}(\lambda)$: This is equivalent to $\mathbf{Hybrid}_{2,\mathsf{Bound}_{\mathcal{A}},2}^{\mathcal{A}}$.
- $\mathbf{Hybrid}_4^{\mathcal{A}}(\lambda)$: Rather than sampling the one-time-pads p_e uniformly at random, we instead sample $\widetilde{x}_e = x_e \oplus p_e$ uniformly at random. This leads to an identical distribution as $\mathbf{Hybrid}_3^{\mathcal{A}}$ as the values of p_e are no longer used since we are simulating all garbled circuits $\widetilde{P}_{f,v}$
- $\mathbf{Hybrid}_5^{\mathcal{A}}(\lambda)$: This is the ideal world experiment written using an explicit simulator Sim. This hybrid is identically distributed to $\mathbf{Hybrid}_4^{\mathcal{A}}$ since the previous hybrid simulates every garbled circuit $\widetilde{P}_{f,v}$ and randomly samples every $\widetilde{x}_e$, and thus does not need to know any input or internal wire values.

5.4.2 Formal Proof.

We now formally prove Theorem 5.1 via a hybrid argument.

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

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs parameters $(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.
2. **Setup:** $K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$. $K_p \leftarrow \mathsf{PRF}_p.\mathsf{Setup}(1^\lambda)$.
3. The adversary can make a polynomial number of function queries and message queries in topological order:

- **Function Query**: On function query $(f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$:
    1. **Compute Keys:**
        (a) For $e \in S_{\mathsf{in}}$, compute $(p_e, \{\mathsf{sk}_{e,k,b}\}_{k\in[\ell_{\mathcal{X}}],b\in\{0,1\}})$ from (K, K_p).
        (b) For $\beta \in S^{\mathsf{int}}_{\mathsf{out}}$, compute $(p_\beta, \{\mathsf{sk}_{\beta,k,b}\}_{k\in[\ell_{\mathcal{X}}],b\in\{0,1\}})$ from (K, K_p).
    2. **Garble Circuit**:
        (a) Let $P_{f,v} = P[f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, \{p_e\}_{e\in S_{\mathsf{in}}}, \{p_\beta, \mathsf{sk}_{\beta,k,b}\}_{\beta\in S^{\mathsf{int}}_{\mathsf{out}},k\in[\ell_{\mathcal{X}}],b\in\{0,1\}}]$ as defined in Figure 4.
        (b) $(\widetilde{P}_{f,v}, \{\mathsf{lab}_{e,k,b}\}_{e\in S_{\mathsf{in}},k\in[\ell_{\mathcal{X}}],b\in\{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, P_{f,v})$.
    3. **Encrypt Labels**:
        (a) For $e \in S_{\mathsf{in}}, k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$, compute $\mathsf{ct}_{e,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{e,k,b}, \mathsf{lab}_{e,k,b})$.
    4. Send $\widehat{F}_{v,S_{\mathsf{in}},S_{\mathsf{out}}} = (\widetilde{P}_{f,v}, \{\mathsf{ct}_{e,k,b}\}_{e\in S_{\mathsf{in}},k\in[\ell_{\mathcal{X}}],b\in\{0,1\}})$ to $\mathcal{A}$.
- **Message Query**: On message query (x_e, e):
    1. Compute $(p_e, \{\mathsf{sk}_{e,k,b}\}_{k\in[\ell_{\mathcal{X}}],b\in\{0,1\}})$ from (K, K_p).
    2. $\widetilde{x}_e = x_e \oplus p_e$.
    3. Send $\widehat{X}_e = (\widetilde{x}_e, \{\mathsf{sk}_{e,k,\widetilde{x}_e[k]}\}_{k\in[\ell_{\mathcal{X}}]})$ to $\mathcal{A}$.

4. **Adversary's Output**: $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$ which is the outcome of the experiment.

Hybrid$_1^{\mathcal{A}}(1^\lambda)$: We now sample the secret keys $\mathsf{sk}_{e,k,b}$ and pads p_e uniformly at random.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs parameters $(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.
2. ~~**Setup:** $K \leftarrow \mathsf{PRF.Setup}(1^\lambda)$. $K_p \leftarrow \mathsf{PRF}_p\mathsf{.Setup}(1^\lambda)$.~~
3. The adversary can make a polynomial number of function queries and message queries in topological order.

- **Function Query**: On function query $(f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$:
  1. **Compute Keys:**
     (a) For $e \in S_{\mathsf{in}}$,
        i. Sample $p_e \leftarrow \{0,1\}^{\ell_{\mathcal{X}}}$ if not already defined.
        ii. Sample $\mathsf{sk}_{e,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$ for $k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$ if not already defined.
     (b) For $\beta \in S_{\mathsf{out}}^{\mathsf{int}}$,
        i. Sample $p_\beta \leftarrow \{0,1\}^{\ell_{\mathcal{X}}}$.
        ii. Sample $\mathsf{sk}_{\beta,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$ for $k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$.
  2. **Garble Circuit**:
     (a) Let $P_{f,v} = P[f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, \{p_e\}_{e \in S_{\mathsf{in}}}, \{p_\beta, \mathsf{sk}_{\beta,k,b}\}_{\beta \in S_{\mathsf{out}}^{\mathsf{int}}, k \in [\ell_{\mathcal{X}}], b \in \{0,1\}}]$ as defined in Figure 4.
     (b) $(\widetilde{P}_{f,v}, \{\mathsf{lab}_{e,k,b}\}_{e \in S_{\mathsf{in}}, k \in [\ell_{\mathcal{X}}], b \in \{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, P_{f,v})$.
  3. **Encrypt Labels**:
     (a) For $e \in S_{\mathsf{in}}, k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$, compute $\mathsf{ct}_{e,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{e,k,b}, \mathsf{lab}_{e,k,b})$.
  4. Send $\widehat{F}_{v,S_{\mathsf{in}},S_{\mathsf{out}}} = (\widetilde{P}_{f,v}, \{\mathsf{ct}_{e,k,b}\}_{e \in S_{\mathsf{in}}, k \in [\ell_{\mathcal{X}}], b \in \{0,1\}})$ to $\mathcal{A}$.
- **Message Query**: On message query (x_e, e):
  1. Sample $p_e \leftarrow \{0,1\}^{\ell_{\mathcal{X}}}$ and $\mathsf{sk}_{e,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$ for $k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$.
  2. $\widetilde{x}_e = x_e \oplus p_e$.
  3. Send $\widehat{X}_e = (\widetilde{x}_e, \{\mathsf{sk}_{e,k,\widetilde{x}_e[k]}\}_{k \in [\ell_{\mathcal{X}}]})$ to $\mathcal{A}$.
4. **Adversary's Output**: $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$ which is the outcome of the experiment.

Lemma 5.5. *If* PRF *and* PRF_p *are secure pseudorandom functions, then for all PPT adversaries* $\mathcal{A}$,

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

Proof. This follow by a straightforward reduction to the security of the PRFs. □

We now go through a sequence of hybrids for $t \in [\mathsf{Bound}_{\mathcal{A}}]$ where $\mathsf{Bound}_{\mathcal{A}}$ is a bound on the size of $\mathcal{A}$, and thus an implicit bound on the number of function and message queries produced by $\mathcal{A}$.

Hybrid$^{\mathcal{A}}_{2,t,0}(1^\lambda)$: We simulate the garbled circuits for all vertices $v < t$. Additionally for all $v < t$, we change the encryptions of labels not corresponding to the correct incoming wire values to encryptions of $\bot$.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs parameters $(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.
2. The adversary can make a polynomial number of function queries and message queries in topological order.

- **Function Query**: On function query $(f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$:
  1. **Compute Keys:**
     (a) For $e \in S_{\mathsf{in}}$,
        i. Sample $p_e \leftarrow \{0,1\}^{\ell_{\mathcal{X}}}$ if not already defined.
        ii. Sample $\mathsf{sk}_{e,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$ for $k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$ if not already defined.
     (b) For $\beta \in S^{\mathsf{int}}_{\mathsf{out}}$,
        i. Sample $p_\beta \leftarrow \{0,1\}^{\ell_{\mathcal{X}}}$.
        ii. Sample $\mathsf{sk}_{\beta,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$ for $k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$.
  2. **Compute Output:**
     (a) For $e \in S_{\mathsf{in}}$, if x_e is not defined, set $x_e = \bot$.
     (b) $(\{x_\alpha\}_{\alpha \in S^{\mathsf{out}}_{\mathsf{out}}}, \{x_\beta\}_{\beta \in S^{\mathsf{int}}_{\mathsf{out}}}) = f(\{x_e\}_{e \in S_{\mathsf{in}}})$.
     (c) For $\beta \in S^{\mathsf{int}}_{\mathsf{out}}$,
        – $\widetilde{x}_\beta = x_\beta \oplus p_\beta$.
        – $\widehat{X}_\beta = (\widetilde{x}_\beta, \{\mathsf{sk}_{\beta,k,\widetilde{x}_\beta[k]}\}_{k \in [\ell_{\mathcal{X}}]})$.
  3. **Garble Circuit**:
     (a) If $v < t$,
        i. $(\widetilde{P}_{f,v}, \{\mathsf{lab}_{e,k,\widetilde{x}[k]}\}_{e \in S_{\mathsf{in}}, k \in [\ell_{\mathcal{X}}]}) \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{\ell_P}, \{\widetilde{x}_e\}_{e \in S_{\mathsf{in}}}, (\{x_\alpha\}_{\alpha \in S^{\mathsf{out}}_{\mathsf{out}}}, \{\widehat{X}_\beta\}_{\beta \in S^{\mathsf{int}}_{\mathsf{out}}}))$.
     (b) Else,
        i. Let $P_{f,v} = P[f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, \{p_e\}_{e \in S_{\mathsf{in}}}, \{p_\beta, \mathsf{sk}_{\beta,k,b}\}_{\beta \in S^{\mathsf{int}}_{\mathsf{out}}, k \in [\ell_{\mathcal{X}}], b \in \{0,1\}}]$ as defined in Figure 4.
        ii. $(\widetilde{P}_{f,v}, \{\mathsf{lab}_{e,k,b}\}_{e \in S_{\mathsf{in}}, k \in [\ell_{\mathcal{X}}], b \in \{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, P_{f,v})$.
  4. **Encrypt Labels**:
     (a) For $e \in S_{\mathsf{in}}, k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$,
        i. If $v < t$ and $\widetilde{x}_e[k] \neq b$, $\mathsf{ct}_{e,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{e,k,b}, \bot)$.
        ii. Else, $\mathsf{ct}_{e,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{e,k,b}, \mathsf{lab}_{e,k,b})$.
  5. Send $\widehat{F}_{v,S_{\mathsf{in}},S_{\mathsf{out}}} = (\widetilde{P}_{f,v}, \{\mathsf{ct}_{e,k,b}\}_{e \in S_{\mathsf{in}}, k \in [\ell_{\mathcal{X}}], b \in \{0,1\}})$ to $\mathcal{A}$.
- **Message Query**: On message query (x_e, e):
  1. Sample $p_e \leftarrow \{0,1\}^{\ell_{\mathcal{X}}}$ and $\mathsf{sk}_{e,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$ for $k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$.
  2. $\widetilde{x}_e = x_e \oplus p_e$.

3. Send $\widehat{X}_e = (\widetilde{x}_e, \{\mathsf{sk}_{e,k,\widetilde{x}_e[k]}\}_{k\in[\ell_{\mathcal{X}}]})$ to $\mathcal{A}$.

3. **Adversary's Output**: $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$ which is the outcome of the experiment.

Lemma 5.6. *For all adversaries* $\mathcal{A}$,

$$\left|\Pr[\mathbf{Hybrid}_1^{\mathcal{A}}(1^\lambda)] - \Pr[\mathbf{Hybrid}_{2,1,0}^{\mathcal{A}}(1^\lambda)]\right| = 0.$$

Proof. The hybrids are identical when $t = 1$. □

Hybrid$^{\mathcal{A}}_{2,t,1}(1^\lambda)$: For vertex v, we change the encryptions of input labels $\{\mathsf{lab}_{e,k,1-\widetilde{x}_e[k]}\}_{e\in S_{\mathsf{in}},k\in[\ell_{\mathcal{X}}]}$ which do not correspond to the wire values $\{\widetilde{x}_e\}_{e\in S_{\mathsf{in}}}$ to encryptions of $\perp$.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs parameters $(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.

2. The adversary can make a polynomial number of function queries and message queries in topological order.

- **Function Query**: On function query $(f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$:

  1. **Compute Keys:**
     (a) For $e \in S_{\mathsf{in}}$,
        i. Sample $p_e \leftarrow \{0,1\}^{\ell_{\mathcal{X}}}$ if not already defined.
        ii. Sample $\mathsf{sk}_{e,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$ for $k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$ if not already defined.
     (b) For $\beta \in S^{\mathsf{int}}_{\mathsf{out}}$,
        i. Sample $p_\beta \leftarrow \{0,1\}^{\ell_{\mathcal{X}}}$.
        ii. Sample $\mathsf{sk}_{\beta,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$ for $k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$.
  2. **Compute Output:**
     (a) For $e \in S_{\mathsf{in}}$, if x_e is not defined, set $x_e = \perp$.
     (b) $(\{x_\alpha\}_{\alpha\in S^{\mathsf{out}}_{\mathsf{out}}}, \{x_\beta\}_{\beta\in S^{\mathsf{int}}_{\mathsf{out}}}) = f(\{x_e\}_{e\in S_{\mathsf{in}}})$.
     (c) For $\beta \in S^{\mathsf{int}}_{\mathsf{out}}$,
        – $\widetilde{x}_\beta = x_\beta \oplus p_\beta$.
        – $\widehat{X}_\beta = (\widetilde{x}_\beta, \{\mathsf{sk}_{\beta,k,\widetilde{x}_\beta[k]}\}_{k\in[\ell_{\mathcal{X}}]})$.
  3. **Garble Circuit**:
     (a) If $v < t$,
        i. $(\widetilde{P}_{f,v}, \{\mathsf{lab}_{e,k,\widetilde{x}[k]}\}_{e\in S_{\mathsf{in}},k\in[\ell_{\mathcal{X}}]}) \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{\ell_P}, \{\widetilde{x}_e\}_{e\in S_{\mathsf{in}}}, (\{x_\alpha\}_{\alpha\in S^{\mathsf{out}}_{\mathsf{out}}}, \{\widehat{X}_\beta\}_{\beta\in S^{\mathsf{int}}_{\mathsf{out}}}))$.
     (b) Else,
        i. Let $P_{f,v} = P[f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, \{p_e\}_{e\in S_{\mathsf{in}}}, \{p_\beta, \mathsf{sk}_{\beta,k,b}\}_{\beta\in S^{\mathsf{int}}_{\mathsf{out}},k\in[\ell_{\mathcal{X}}],b\in\{0,1\}}]$ as defined in Figure 4.
        ii. $(\widetilde{P}_{f,v}, \{\mathsf{lab}_{e,k,b}\}_{e\in S_{\mathsf{in}},k\in[\ell_{\mathcal{X}}],b\in\{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, P_{f,v})$.
  4. **Encrypt Labels**:
     (a) For $e \in S_{\mathsf{in}}, k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$,
        i. If $v \le t$ and $\widetilde{x}_e[k] \neq b$, $\mathsf{ct}_{e,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{e,k,b}, \perp)$.
        ii. Else, $\mathsf{ct}_{e,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{e,k,b}, \mathsf{lab}_{e,k,b})$.
  5. Send $\widehat{F}_{v,S_{\mathsf{in}},S_{\mathsf{out}}} = (\widetilde{P}_{f,v}, \{\mathsf{ct}_{e,k,b}\}_{e\in S_{\mathsf{in}},k\in[\ell_{\mathcal{X}}],b\in\{0,1\}})$ to $\mathcal{A}$.

- **Message Query**: On message query (x_e, e):

  1. Sample $p_e \leftarrow \{0,1\}^{\ell_{\mathcal{X}}}$ and $\mathsf{sk}_{e,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$ for $k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$.
  2. $\widetilde{x}_e = x_e \oplus p_e$.
  3. Send $\widehat{X}_e = (\widetilde{x}_e, \{\mathsf{sk}_{e,k,\widetilde{x}_e[k]}\}_{k\in[\ell_{\mathcal{X}}]})$ to $\mathcal{A}$.

3. **Adversary's Output**: $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$ which is the outcome of the experiment.

Lemma 5.7. *If* SKE *is a secret key encryption scheme, then for all PPT adversaries* $\mathcal{A}$ *and all* $t \in \mathbb{N}$*,*

$$\left|\Pr[\mathbf{Hybrid}^{\mathcal{A}}_{2,t,0}(1^\lambda)] - \Pr[\mathbf{Hybrid}^{\mathcal{A}}_{2,t,1}(1^\lambda)]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Consider an incoming edge $e \in S_{\mathsf{in}}$ to vertex $v = t$. Since the queries must occur in topological order and the vertex labels occur in increasing order (see Remarks 5.3 and 5.4), this edge e can only occur in the hybrid as (i) a message query, (ii) as an outgoing edge for some vertex $v' < t$, and/or (iii) as an incoming edge for vertex $v = t$ since $e \in S_{\mathsf{in}}$. We observe that the third case is the only place where the secret keys $\{\mathsf{sk}_{e,k,1-\widetilde{x}_e}[k]\}_{k\in[\ell_{\mathcal{X}}]}$, which do *not* correspond to the computed value $\widetilde{x}_e$, are used. In the first case of a message query, we only use secret keys $\{\mathsf{sk}_{e,k,\widetilde{x}_e}[k]\}_{k\in[\ell_{\mathcal{X}}]}$ corresponding to the one-time-padded message $\widetilde{x}_e$. In the second case, since we simulate all garbled circuits for $v' < t$, then the function queries for $v' < t$ also only need to make use of secret keys $\{\mathsf{sk}_{e,k,\widetilde{x}_e}[k]\}_{k\in[\ell_{\mathcal{X}}]}$ corresponding to the one-time-padded message $\widetilde{x}_e$. Thus, the only place the keys $\{\mathsf{sk}_{e,k,1-\widetilde{x}_e}[k]\}_{k\in[\ell_{\mathcal{X}}]}$ are used in this and the previous hybrid are during the function query for $v = t$. In the previous hybrid, the function query for $v = t$ uses these keys to encrypt the corresponding garbled circuit labels $\{\mathsf{lab}_{e,k,1-\widetilde{x}_e}[k]\}_{k\in[\ell_{\mathcal{X}}]}$, and in the current hybrid, they are used to encrypt $\perp$. Therefore, the lemma follows from a straightforward reduction to the security of the SKE scheme since we are simply swapping encryptions of $\{\mathsf{lab}_{e,k,1-\widetilde{x}_e}[k]\}_{k\in[\ell_{\mathcal{X}}]}$ to encryptions of $\perp$. □

Hybrid$^{\mathcal{A}}_{2,t,2}(1^\lambda)$: We simulate the garbled circuit for vertex $v = t$.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs parameters $(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.
2. The adversary can make a polynomial number of function queries and message queries in topological order.

- **Function Query**: On function query $(f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$:
  1. **Compute Keys:**
     (a) For $e \in S_{\mathsf{in}}$,
        i. Sample $p_e \leftarrow \{0,1\}^{\ell_{\mathcal{X}}}$ if not already defined.
        ii. Sample $\mathsf{sk}_{e,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$ for $k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$ if not already defined.
     (b) For $\beta \in S^{\mathsf{int}}_{\mathsf{out}}$,
        i. Sample $p_\beta \leftarrow \{0,1\}^{\ell_{\mathcal{X}}}$.
        ii. Sample $\mathsf{sk}_{\beta,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$ for $k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$.
  2. **Compute Output:**
     (a) For $e \in S_{\mathsf{in}}$, if x_e is not defined, set $x_e = \bot$.
     (b) $(\{x_\alpha\}_{\alpha \in S^{\mathsf{out}}_{\mathsf{out}}}, \{x_\beta\}_{\beta \in S^{\mathsf{int}}_{\mathsf{out}}}) = f(\{x_e\}_{e \in S_{\mathsf{in}}})$.
     (c) For $\beta \in S^{\mathsf{int}}_{\mathsf{out}}$,
        – $\widetilde{x}_\beta = x_\beta \oplus p_\beta$.
        – $\widehat{X}_\beta = (\widetilde{x}_\beta, \{\mathsf{sk}_{\beta,k,\widetilde{x}_\beta[k]}\}_{k \in [\ell_{\mathcal{X}}]})$.
  3. **Garble Circuit**:
     (a) If $v \leq t$,
        i. $(\widetilde{P}_{f,v}, \{\mathsf{lab}_{e,k,\widetilde{x}[k]}\}_{e \in S_{\mathsf{in}}, k \in [\ell_{\mathcal{X}}]}) \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{\ell_P}, \{\widetilde{x}_e\}_{e \in S_{\mathsf{in}}}, (\{x_\alpha\}_{\alpha \in S^{\mathsf{out}}_{\mathsf{out}}}, \{\widehat{X}_\beta\}_{\beta \in S^{\mathsf{int}}_{\mathsf{out}}}))$.
     (b) Else,
        i. Let $P_{f,v} = P[f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, \{p_e\}_{e \in S_{\mathsf{in}}}, \{p_\beta, \mathsf{sk}_{\beta,k,b}\}_{\beta \in S^{\mathsf{int}}_{\mathsf{out}}, k \in [\ell_{\mathcal{X}}], b \in \{0,1\}}]$ as defined in Figure 4.
        ii. $(\widetilde{P}_{f,v}, \{\mathsf{lab}_{e,k,b}\}_{e \in S_{\mathsf{in}}, k \in [\ell_{\mathcal{X}}], b \in \{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, P_{f,v})$.
  4. **Encrypt Labels**:
     (a) For $e \in S_{\mathsf{in}}, k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$,
        i. If $v \leq t$ and $\widetilde{x}_e[k] \neq b$, $\mathsf{ct}_{e,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{e,k,b}, \bot)$.
        ii. Else, $\mathsf{ct}_{e,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{e,k,b}, \mathsf{lab}_{e,k,b})$.
  5. Send $\widehat{F}_{v,S_{\mathsf{in}},S_{\mathsf{out}}} = (\widetilde{P}_{f,v}, \{\mathsf{ct}_{e,k,b}\}_{e \in S_{\mathsf{in}}, k \in [\ell_{\mathcal{X}}], b \in \{0,1\}})$ to $\mathcal{A}$.
- **Message Query**: On message query (x_e, e):
  1. Sample $p_e \leftarrow \{0,1\}^{\ell_{\mathcal{X}}}$ and $\mathsf{sk}_{e,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$ for $k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$.
  2. $\widetilde{x}_e = x_e \oplus p_e$.
  3. Send $\widehat{X}_e = (\widetilde{x}_e, \{\mathsf{sk}_{e,k,\widetilde{x}_e[k]}\}_{k \in [\ell_{\mathcal{X}}]})$ to $\mathcal{A}$.

3. **Adversary's Output**: $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$ which is the outcome of the experiment.

Lemma 5.8. *If* GC *is a secure garbling scheme, then for all PPT adversaries* $\mathcal{A}$ *and all* $t \in \mathbb{N}$*,*

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

Proof. The lemma follows from a straightforward reduction to the security of the garbled circuit scheme. Observe that since the queries are given in topological order, the input labels $\{\mathsf{lab}_{e,k,1-\widetilde{x}_e}[k]\}_{e\in S_{\mathsf{in}},k\in[\ell_{\mathcal{X}}]}$ which do not correspond to the input values $\{\widetilde{x}_e\}_{e\in S_{\mathsf{in}}}$ for vertex v are not used in the hybrid. □

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

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

Proof. The hybrids are identical. □

$\mathbf{Hybrid}_3^{\mathcal{A}}(1^\lambda)$: This is equivalent to $\mathbf{Hybrid}_{2,\mathsf{Bound}_{\mathcal{A}},2}^{\mathcal{A}}$ where $\mathsf{Bound}_{\mathcal{A}}$ is an upper bound on the size of the adversary $\mathcal{A}$, and thus an upper bound on the number of queries the adversary can make.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs parameters $(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.

2. The adversary can make a polynomial number of function queries and message queries in topological order.

- **Function Query**: On function query $(f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$:

  1. **Compute Keys:**
     (a) For $e \in S_{\mathsf{in}}$,
        i. Sample $p_e \leftarrow \{0,1\}^{\ell_{\mathcal{X}}}$ if not already defined.
        ii. Sample $\mathsf{sk}_{e,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$ for $k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$ if not already defined.
     (b) For $\beta \in S_{\mathsf{out}}^{\mathsf{int}}$,
        i. Sample $p_\beta \leftarrow \{0,1\}^{\ell_{\mathcal{X}}}$.
        ii. Sample $\mathsf{sk}_{\beta,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$ for $k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$.
  2. **Compute Output:**
     (a) For $e \in S_{\mathsf{in}}$, if x_e is not defined, set $x_e = \bot$.
     (b) $(\{x_\alpha\}_{\alpha \in S_{\mathsf{out}}^{\mathsf{out}}}, \{x_\beta\}_{\beta \in S_{\mathsf{out}}^{\mathsf{int}}}) = f(\{x_e\}_{e \in S_{\mathsf{in}}})$.
     (c) For $\beta \in S_{\mathsf{out}}^{\mathsf{int}}$,
        – $\widetilde{x}_\beta = x_\beta \oplus p_\beta$.
        – $\widehat{X}_\beta = (\widetilde{x}_\beta, \{\mathsf{sk}_{\beta,k,\widetilde{x}_\beta[k]}\}_{k \in [\ell_{\mathcal{X}}]})$.
  3. **Garble Circuit**:
     (a) $(\widetilde{P}_{f,v}, \{\mathsf{lab}_{e,k,\widetilde{x}[k]}\}_{e \in S_{\mathsf{in}}, k \in [\ell_{\mathcal{X}}]}) \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{\ell_P}, \{\widetilde{x}_e\}_{e \in S_{\mathsf{in}}}, (\{x_\alpha\}_{\alpha \in S_{\mathsf{out}}^{\mathsf{out}}}, \{\widehat{X}_\beta\}_{\beta \in S_{\mathsf{out}}^{\mathsf{int}}}))$..
  4. **Encrypt Labels**:
     (a) For $e \in S_{\mathsf{in}}, k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$,
        i. If $\widetilde{x}_e[k] \neq b$, $\mathsf{ct}_{e,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{e,k,b}, \bot)$.
        ii. Else, $\mathsf{ct}_{e,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{e,k,b}, \mathsf{lab}_{e,k,b})$.
  5. Send $\widehat{F}_{v,S_{\mathsf{in}},S_{\mathsf{out}}} = (\widetilde{P}_{f,v}, \{\mathsf{ct}_{e,k,b}\}_{e \in S_{\mathsf{in}}, k \in [\ell_{\mathcal{X}}], b \in \{0,1\}})$ to $\mathcal{A}$.

- **Message Query**: On message query (x_e, e):

  1. Sample $p_e \leftarrow \{0,1\}^{\ell_{\mathcal{X}}}$ and $\mathsf{sk}_{e,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$ for $k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$.
  2. $\widetilde{x}_e = x_e \oplus p_e$.
  3. Send $\widehat{X}_e = (\widetilde{x}_e, \{\mathsf{sk}_{e,k,\widetilde{x}_e[k]}\}_{k \in [\ell_{\mathcal{X}}]})$ to $\mathcal{A}$.

3. **Adversary's Output**: $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$ which is the outcome of the experiment.

Lemma 5.10. *For all adversaries $\mathcal{A}$ of size at most $\mathsf{Bound}_{\mathcal{A}}$,*

$$\left|\Pr[\mathbf{Hybrid}_{2,\mathsf{Bound}_{\mathcal{A}},2}^{\mathcal{A}}(1^\lambda)] - \Pr[\mathbf{Hybrid}_3^{\mathcal{A}}(1^\lambda)]\right| = 0.$$

Proof. The hybrids are identical since the adversary can make at most $\mathsf{Bound}_{\mathcal{A}}$ queries. □

Hybrid$_4^{\mathcal{A}}(1^\lambda)$: Now, rather than sampling the one-time-pads p_e uniformly at random, we instead sample $\widetilde{x}_e = x_e \oplus p_e$ uniformly at random.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs parameters $(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.
2. The adversary can make a polynomial number of function queries and message queries in topological order.

- **Function Query**: On function query $(f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$:
  1. **Compute Keys:**
     (a) For $e \in S_{\mathsf{in}}$,
        i. ~~Sample $p_e \leftarrow \{0,1\}^{\ell_{\mathcal{X}}}$ if not already defined.~~
        ii. Sample $\mathsf{sk}_{e,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$ for $k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$ if not already defined.
     (b) For $\beta \in S_{\mathsf{out}}^{\mathsf{int}}$,
        i. ~~Sample $p_\beta \leftarrow \{0,1\}^{\ell_{\mathcal{X}}}$.~~
        ii. Sample $\mathsf{sk}_{\beta,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$ for $k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$.
  2. **Compute Output:**
     (a) For $e \in S_{\mathsf{in}}$, if x_e is not defined, set $x_e = \bot$.
     (b) $(\{x_\alpha\}_{\alpha \in S_{\mathsf{out}}^{\mathsf{out}}}, \{x_\beta\}_{\beta \in S_{\mathsf{out}}^{\mathsf{int}}}) = f(\{x_e\}_{e \in S_{\mathsf{in}}})$.
     (c) For $\beta \in S_{\mathsf{out}}^{\mathsf{int}}$,
        – Sample $\widetilde{x}_\beta \leftarrow \{0,1\}^{\ell_{\mathcal{X}}}$.
        – $\widehat{X}_\beta = (\widetilde{x}_\beta, \{\mathsf{sk}_{\beta,k,\widetilde{x}_\beta[k]}\}_{k \in [\ell_{\mathcal{X}}]})$.
  3. **Garble Circuit**:
     (a) $(\widetilde{P}_{f,v}, \{\mathsf{lab}_{e,k,\widetilde{x}[k]}\}_{e \in S_{\mathsf{in}}, k \in [\ell_{\mathcal{X}}]}) \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{\ell_P}, \{\widetilde{x}_e\}_{e \in S_{\mathsf{in}}}, (\{x_\alpha\}_{\alpha \in S_{\mathsf{out}}^{\mathsf{out}}}, \{\widehat{X}_\beta\}_{\beta \in S_{\mathsf{out}}^{\mathsf{int}}}))$..
  4. **Encrypt Labels**:
     (a) For $e \in S_{\mathsf{in}}, k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$,
        i. If $\widetilde{x}_e[k] \neq b$, $\mathsf{ct}_{e,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{e,k,b}, \bot)$.
        ii. Else, $\mathsf{ct}_{e,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{e,k,b}, \mathsf{lab}_{e,k,b})$.
  5. Send $\widehat{F}_{v,S_{\mathsf{in}},S_{\mathsf{out}}} = (\widetilde{P}_{f,v}, \{\mathsf{ct}_{e,k,b}\}_{e \in S_{\mathsf{in}}, k \in [\ell_{\mathcal{X}}], b \in \{0,1\}})$ to $\mathcal{A}$.
- **Message Query**: On message query (x_e, e):
  1. Sample ~~$p_e \leftarrow \{0,1\}^{\ell_{\mathcal{X}}}$~~ and $\mathsf{sk}_{e,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$ for $k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$.
  2. Sample $\widetilde{x}_e \leftarrow \{0,1\}^{\ell_{\mathcal{X}}}$.
  3. Send $\widehat{X}_e = (\widetilde{x}_e, \{\mathsf{sk}_{e,k,\widetilde{x}_e[k]}\}_{k \in [\ell_{\mathcal{X}}]})$ to $\mathcal{A}$.
3. **Adversary's Output**: $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$ which is the outcome of the experiment.

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

$$\left|\Pr[\mathbf{Hybrid}_3^{\mathcal{A}}(1^\lambda)] - \Pr[\mathbf{Hybrid}_4^{\mathcal{A}}(1^\lambda)]\right| = 0.$$

Proof. The hybrids are identically distributed. Note that we no longer need any of the p_e values since we are simulating all of our garbled circuits $\widetilde{P}_{f,v}$. □

Hybrid$_5^{\mathcal{A}}(1^\lambda)$: This is the ideal world written using an explicit simulator Sim. This hybrid is identical to the previous hybrid except that we now get the output values $\{x_\alpha\}_{\alpha\in S_{\mathsf{out}}^{\mathsf{out}}}$ from the security game.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs parameters $(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.
2. **Setup**: The simulator Sim receives $(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$ which it stores for later use.

- **Function Query**: On function query $(\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$,
  We compute the following output values:

  1. For $e \in S_{\mathsf{in}}$, if x_e is not already defined, set $x_e = \bot$.
  2. $(\{x_\alpha\}_{\alpha\in S_{\mathsf{out}}^{\mathsf{out}}}, \{x_\beta\}_{\beta\in S_{\mathsf{out}}^{\mathsf{int}}}) = f(\{x_e\}_{e\in S_{\mathsf{in}}})$.

  The simulator Sim then receives $(\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, \{x_\alpha\}_{\alpha\in S_{\mathsf{out}}^{\mathsf{out}}})$ and does the following:

  1. **Compute Keys:**
     (a) For $e \in S_{\mathsf{in}}$,
        i. Sample $\mathsf{sk}_{e,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$ for $k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$ if not already defined.
     (b) For $\beta \in S_{\mathsf{out}}^{\mathsf{int}}$,
        i. Sample $\mathsf{sk}_{\beta,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$ for $k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$.
  2. **Compute Output:**
     (a) For $\beta \in S_{\mathsf{out}}^{\mathsf{int}}$,
        – Sample $\widetilde{x}_\beta \leftarrow \{0,1\}^{\ell_{\mathcal{X}}}$.
        – $\widehat{X}_\beta = (\widetilde{x}_\beta, \{\mathsf{sk}_{\beta,k,\widetilde{x}_\beta[k]}\}_{k\in[\ell_{\mathcal{X}}]})$.
  3. **Garble Circuit**:
     (a) $(\widetilde{P}_{f,v}, \{\mathsf{lab}_{e,k,\widetilde{x}[k]}\}_{e\in S_{\mathsf{in}},k\in[\ell_{\mathcal{X}}]}) \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{\ell_P}, \{\widetilde{x}_e\}_{e\in S_{\mathsf{in}}}, (\{x_\alpha\}_{\alpha\in S_{\mathsf{out}}^{\mathsf{out}}}, \{\widehat{X}_\beta\}_{\beta\in S_{\mathsf{out}}^{\mathsf{int}}}))$..
  4. **Encrypt Labels**:
     (a) For $e \in S_{\mathsf{in}}, k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$,
        i. If $\widetilde{x}_e[k] \neq b$, $\mathsf{ct}_{e,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{e,k,b}, \bot)$.
        ii. Else, $\mathsf{ct}_{e,k,b} \leftarrow \mathsf{SKE.Enc}(\mathsf{sk}_{e,k,b}, \mathsf{lab}_{e,k,b})$.
  5. Send $\widehat{F}_{v,S_{\mathsf{in}},S_{\mathsf{out}}} = (\widetilde{P}_{f,v}, \{\mathsf{ct}_{e,k,b}\}_{e\in S_{\mathsf{in}},k\in[\ell_{\mathcal{X}}],b\in\{0,1\}})$ to $\mathcal{A}$.

- **Message Query**: On message query (x_e, e),
  The simulator Sim receives just (e) and does the following:

  1. Sample $\mathsf{sk}_{e,k,b} \leftarrow \mathsf{SKE.Setup}(1^\lambda)$ for $k \in [\ell_{\mathcal{X}}], b \in \{0,1\}$.
  2. Sample $\widetilde{x}_e \leftarrow \{0,1\}^{\ell_{\mathcal{X}}}$.
  3. Send $\widehat{X}_e = (\widetilde{x}_e, \{\mathsf{sk}_{e,k,\widetilde{x}_e[k]}\}_{k\in[\ell_{\mathcal{X}}]})$ to $\mathcal{A}$.

3. **Adversary's Output**: $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$ which is the outcome of the experiment.

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

$$\left|\Pr[\mathbf{Hybrid}_4^{\mathcal{A}}(1^\lambda)] - \Pr[\mathbf{Hybrid}_5^{\mathcal{A}}(1^\lambda)]\right| = 0.$$

Proof. The hybrids are identical. □

Thus, our lemmas give us the following corollary:

Corollary 5.13. *If*

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

then MHGC *is a secure multi-hop garbling scheme for* $\mathsf{P/Poly}$.

Proof. The corollary immediately follows from Lemmas 5.5-5.12. □

Corollary 5.13 then implies Theorem 5.1 since we can instantiate each of the required primitives from OWFs.

6 Bootstrapping to a Multi-Hop Functional Encryption Scheme

In this section, we prove the following theorem:

Theorem 6.1. *Assuming an adaptive-IND secure public-key* FE *scheme for* $\mathsf{P/Poly}$*, there exists a one-ciphertext, SIM-secure public-key* MHFE *scheme for* $\mathsf{P/Poly}$.

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

Tools.

- $\mathsf{MHGC} = (\mathsf{MHGC.Setup}, \mathsf{MHGC.GarbInp}, \mathsf{MHGC.GarbCirc}, \mathsf{MHGC.Eval})$: A secure multi-hop garbled circuit scheme for $\mathsf{P/Poly}$.
- $\mathsf{PRF}_1, \mathsf{PRF}_2, \mathsf{PRF}_3$ where $\mathsf{PRF}_c = (\mathsf{PRF}_c.\mathsf{Setup}, \mathsf{PRF}_c.\mathsf{Eval})$ for $c \in [3]$: secure pseudorandom function families.
- $\mathsf{SKE}_1, \mathsf{SKE}_2$ where $\mathsf{SKE}_c = \mathsf{SKE}_c.\mathsf{Setup}, \mathsf{SKE}_c.\mathsf{Enc}, \mathsf{SKE}_c.\mathsf{Dec})$ for $c \in [2]$: secure secret key encryption schemes with pseudorandom ciphertexts.
- $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.Enc}, \mathsf{FE.KeyGen}, \mathsf{FE.Dec})$: An adaptive-IND-secure public-key FE scheme for $\mathsf{P/Poly}$.
- $\mathsf{FPFE} = (\mathsf{FPFE.Setup}, \mathsf{FPFE.Enc}, \mathsf{FPFE.KeyGen}, \mathsf{FPFE.Dec})$: A function-private, adaptive-IND-secure secret-key FE scheme for $\mathsf{P/Poly}$

Instantiation of the Tools. Let FE be an adaptive-IND-secure public-key FE scheme for $\mathsf{P/Poly}$.

- We can build $\mathsf{PRF}_1, \mathsf{PRF}_2, \mathsf{PRF}_3, \mathsf{SKE}_1, \mathsf{SKE}_2$ from any OWF using standard cryptographic techniques (e.g. [Gol01, Gol09]). As FE implies OWFs, then we can build these from FE.
- Theorem 5.1 shows how to build a secure MHGC scheme for $\mathsf{P/Poly}$ from OWFs. As FE implies OWFs, then we can build these from FE.
- FE immediately implies a an adaptive-IND-secure, *secret-key* FE scheme SKFE for $\mathsf{P/Poly}$. We can then build our function-private, adaptive-IND-secure secret-key FE scheme FPFE for $\mathsf{P/Poly}$ by using the function-privacy transformation of [BS18] on SKFE.

6.1 Parameters

On security parameter λ, maximum degree d_{max}, function size $\ell_{\mathcal{F}}$, message size $\ell_{\mathcal{X}}$, and type function type, we instantiate our primitives with the following parameters:

- We instantiate MHGC with security parameter λ, maximum degree d_{max}, function size $\ell_{\mathcal{F}}$, message size $\ell_{\mathcal{X}}$, and type function type.
- We instantiate our other primitives according to the following table:

	Security Parameter	Input Size	Output Size	Function Size
PRF_1	λ	λ	2λ	
PRF_2	λ	2λ	2λ	
PRF_3	λ	λ	λ	
SKE_2	λ	$\ell_{H.out}$		
FPFE	λ	$\ell_{H.in}$	$\ell_{H.out}$	ℓ_H
SKE_1	λ	$\ell_{G.out}$		
FE	λ	$\ell_{G.in}$	$\ell_{G.out}$	ℓ_G

where we define

- $\ell_{H.in} = \ell_{\mathcal{X}} + \lambda + 1 + \ell_{\mathsf{SKE}_2.\mathsf{sk}}$, where $\ell_{\mathsf{SKE}_2.\mathsf{sk}}$ is the size of SKE_2 secret keys.
  (This is also the size of inputs to H defined in Figure 6).
- $\ell_{H.out} = \ell_{\mathsf{MHGC}.\widehat{X}}$ is the size of MHGC garbled inputs.
  (This is also the size of inputs to H defined in Figure 6).
- ℓ_H is the maximum size of any $H_{\mathsf{tid},v}$ defined in Figure 6 or $H^*_{\mathsf{tid},v}$ defined in Figure 7 for any $\mathsf{tid} \in \{0,1\}^\lambda, v \in \{0,1\}^\lambda, S_{\mathsf{in}} \subseteq \{0,1\}^\lambda$ where $|S_{\mathsf{in}}| \leq d_{\mathsf{max}}$, MHGC master secret key $\mathsf{MHGC.msk}_{\mathsf{tid}}$, PRF_3 key $\mathsf{PRF}_3.K_{\mathsf{tid}}$, SKE_2 ciphertexts $\{\mathsf{ct}_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}}$, $x_e \in \{0,1\}^{\ell_{\mathcal{X}}}, e \in \{0,1\}^\lambda, \mathsf{branch}_2 \in \{0,1\}$, and SKE_2 secret key $\mathsf{SKE}_2.\mathsf{sk}$.
- $\ell_{G.in} = \ell_{\mathsf{FPFE.msk}} + \ell_{\mathsf{PRF}_1.K} + \ell_{\mathsf{PRF}_2.K} + 1 + \ell_{\mathsf{SKE}_1.\mathsf{sk}}$, where $\ell_{\mathsf{FPFE.msk}}$ is the size of FPFE master secret keys, $\ell_{\mathsf{PRF}_1.K}$ is the size of PRF_1 keys, $\ell_{\mathsf{PRF}_2.K}$ is the size of PRF_2 keys, and $\ell_{\mathsf{SKE}_1.\mathsf{sk}}$ is the size of SKE_1 keys.
  (This is also the size of inputs to G defined in Figure 5).
- $\ell_{G.out} = \ell_{\mathsf{MHGC}.\widehat{F}} + \ell_{\mathsf{FPFE.sk}}$, where $\ell_{\mathsf{MHGC}.\widehat{F}}$ is the size of MHGC garbled circuits, and $\ell_{\mathsf{FPFE.sk}}$ is the size of FPFE function keys.
  (This is also the size of outputs of G defined in Figure 5).

– ℓ_H is the maximum size of any $G_{\mathsf{tid},v}$ defined in Figure 5 for any $\mathsf{tid} \in \{0,1\}^\lambda, f \in \{0,1\}^{\ell_\mathcal{F}}, v \in \{0,1\}^\lambda, S_{\mathsf{in}}, S_{\mathsf{out}} \subseteq \{0,1\}^\lambda$ where $|S_{\mathsf{in}}|, |S_{\mathsf{out}}| \leq d_{\mathsf{max}}$, SKE_1 ciphertext $c_{\mathsf{tid},v}$, SKE_2 ciphertexts $\{c_{\mathsf{tid},e}\}_{e\ in S_{\mathsf{in}}}$, FPFE master secret key $\mathsf{FPFE.msk}$, PRF_1 key $\mathsf{PRF}_1.K$, PRF_2 key $\mathsf{PRF}_2.K$, $\mathsf{branch}_1 \in \{0,1\}$, and SKE_1 secret key $\mathsf{SKE}_1.\mathsf{sk}$.

Remark 6.2. We assume without loss of generality that for security parameter λ, all algorithms only require randomness of length λ. If the original algorithm required additional randomness, we can replace it with a new algorithm that first expands the λ bits of randomness using a PRG of appropriate stretch and then runs the original algorithm. Note that this replacement can be implemented with OWFs and does not affect the security of the above schemes (as long as $d_{\mathsf{max}}, \ell_\mathcal{F}, \ell_\mathcal{X}, |\mathsf{type}|$ are polynomial in λ).

6.2 Construction

We now construct one-ciphertext, SIM-secure public-key MHFE for $\mathsf{P/Poly}$.

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

Notation. Recall that if S is any set of edges and we have an edge labeling function $\mathsf{type} : S \to \{\mathsf{input}, \mathsf{output}, \mathsf{internal}\}$, then we define $S^{\mathsf{inp}}, S^{\mathsf{out}}, S^{\mathsf{int}}$ to be the restrictions of S to edges with type $\mathsf{input}, \mathsf{output}, \mathsf{internal}$ respectively, i.e. $S^{\mathsf{inp}} := \{e \in S : \mathsf{type}(e) = \mathsf{input}\}$, $S^{\mathsf{out}} := \{e \in S : \mathsf{type}(e) = \mathsf{output}\}$, $S^{\mathsf{int}} := \{e \in S : \mathsf{type}(e) = \mathsf{internal}\}$.

MHFE **Construction:**

- $\mathsf{MHFE.Setup}(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$:
  1. $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
  2. Output $(\mathsf{MPK}, \mathsf{MSK}) = (\mathsf{FE.mpk}, \mathsf{FE.msk})$.
- $\mathsf{MHFE.EncSetup}(\mathsf{MPK})$:
  1. Parse $\mathsf{MPK} = \mathsf{FE.mpk}$.
  2. $\mathsf{PRF}_1.K \leftarrow \mathsf{PRF}_1.\mathsf{Setup}(1^\lambda)$.
  3. $\mathsf{PRF}_2.K \leftarrow \mathsf{PRF}_2.\mathsf{Setup}(1^\lambda)$.
  4. $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$.
  5. $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (\mathsf{FPFE.msk}, \mathsf{PRF}_1.K, \mathsf{PRF}_2.K, 0, 0^{\ell_{\mathsf{SKE}_1 \cdot k}}))$.
  6. Output $\mathsf{EK} = (\mathsf{FPFE.msk}, \mathsf{FE.ct})$.
- $\mathsf{MHFE.Enc}(\mathsf{EK}, x_e, e)$:
  1. Parse $\mathsf{EK} = (\mathsf{FPFE.msk}, \mathsf{FE.ct})$.
  2. $\mathsf{FPFE.ct}_e \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (x_e, e, 0, 0^{\ell_{\mathsf{SKE}_2 \cdot k}}))$.
  3. Output $\mathsf{CT}_e = (\mathsf{FE.ct}, \mathsf{FPFE.ct}_e)$. (For all tid, we define $\mathsf{CT}_{\mathsf{tid},e} = \mathsf{CT}_e$.)
- $\mathsf{MHFE.KeyGen}(\mathsf{MSK}, \mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$:
  1. Parse $\mathsf{MSK} = \mathsf{FE.msk}$.
  2. Sample $c_{\mathsf{tid},v} \leftarrow \{0,1\}^{\ell_{\mathsf{SKE.ct}}}$.
  3. For $e \in S_{\mathsf{in}}$, sample $c_{\mathsf{tid},e} \leftarrow \{0,1\}^{\ell_{\mathsf{SKE.ct}}}$.
  4. Let $G_{\mathsf{tid},v} = G[\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, c_{\mathsf{tid},v}, \{c_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}}]$ as defined in Figure 5.
  5. $\mathsf{FE.sk}_{G_{\mathsf{tid},v}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_{\mathsf{tid},v})$.
  6. Output $\mathsf{SK}_{\mathsf{tid},v} = \mathsf{FE.sk}_{G_{\mathsf{tid},v}}$.

$G[\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, c_{\mathsf{tid},v}, \{c_{\mathsf{tid},e}\}_{e\in S_{\mathsf{in}}}](\mathsf{FPFE.msk}, \mathsf{PRF}_1.K, \mathsf{PRF}_2.K, \mathsf{branch}_1, \mathsf{SKE}_1.\mathsf{sk})$:

1. If $\mathsf{branch}_1 = 0$,
   (a) $(r_{\mathsf{Setup}}, r_{\mathsf{PRF}}) \leftarrow \mathsf{PRF}_1.\mathsf{Eval}(\mathsf{PRF}_1.K, \mathsf{tid})$.
   (b) $\mathsf{MHGC.msk}_{\mathsf{tid}} \leftarrow \mathsf{MHGC.Setup}(1^\lambda; r_{\mathsf{Setup}})$.
   (c) $\mathsf{PRF}_3.K_{\mathsf{tid}} \leftarrow \mathsf{PRF}_3.\mathsf{Setup}(1^\lambda; r_{\mathsf{PRF}})$.
   (d) $(r_H, r_{\mathsf{GarbCirc}}) \leftarrow \mathsf{PRF}_2.\mathsf{Eval}(\mathsf{PRF}_2.K, (\mathsf{tid}, v))$.
   (e) Let $H_{\mathsf{tid},v} = H[\mathsf{tid}, v, S_{\mathsf{in}}, \mathsf{MHGC.msk}_{\mathsf{tid}}, \mathsf{PRF}_3.K_{\mathsf{tid}}, \{c_{\mathsf{tid},e}\}_{e\in S_{\mathsf{in}}}]$ as defined in Figure 6.
   (f) $\mathsf{FPFE.sk}_{H_{\mathsf{tid},v}} \leftarrow \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H_{\mathsf{tid},v}; r_H)$.
   (g) $\widehat{F}_{\mathsf{tid},v} \leftarrow \mathsf{MHGC.GarbCirc}(\mathsf{MHGC.msk}_{\mathsf{tid}}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}}; r_{\mathsf{GarbCirc}})$.
   (h) Output $(\widehat{F}_{\mathsf{tid},v}, \mathsf{FPFE.sk}_{H_{\mathsf{tid},v}})$.
2. Else,
   (a) Output $(\widehat{F}_{\mathsf{tid},v}, \mathsf{FPFE.sk}_{H_{\mathsf{tid},v}}) \leftarrow \mathsf{SKE}_1.\mathsf{Dec}(\mathsf{SKE}_1.\mathsf{sk}, c_{\mathsf{tid},v})$.

Figure 5: Definition of $G_{\mathsf{tid},v}$

$H[\mathsf{tid}, v, S_{\mathsf{in}}, \mathsf{MHGC.msk}, \mathsf{PRF}_3.K_{\mathsf{tid}}, \{c_{\mathsf{tid},e}\}_{e\in S_{\mathsf{in}}}](x_e, e, \mathsf{branch}_2, \mathsf{SKE}_2.\mathsf{sk})$:

1. If $e \notin S_{\mathsf{in}}$, output $\perp$.
2. If $\mathsf{branch}_2 = 0$,
   (a) $r_{\mathsf{GarbInp}} \leftarrow \mathsf{PRF}_3.\mathsf{Eval}(\mathsf{PRF}_3.K_{\mathsf{tid}}, e)$.
   (b) Output $\widehat{X}_{\mathsf{tid},e} \leftarrow \mathsf{MHGC.GarbInp}(\mathsf{MHGC.msk}, x_e, e; r_{\mathsf{GarbInp}})$.
3. Else,
   (a) Output $\widehat{X}_{\mathsf{tid},e} \leftarrow \mathsf{SKE}_2.\mathsf{Dec}(\mathsf{SKE}_2.\mathsf{sk}, c_{\mathsf{tid},e})$

Figure 6: Definition of $H_{\mathsf{tid},v}$

- $\mathsf{MHFE.Dec}(\mathsf{SK}_{\mathsf{tid},v}, \{\mathsf{CT}_{\mathsf{tid},e}\}_{e\in S_{\mathsf{in}}})$:
  1. Parse $\mathsf{SK}_{\mathsf{tid},v} = \mathsf{FE.sk}_{G_{\mathsf{tid},v}}$.
  2. For $e \in S_{\mathsf{in}}^{\mathsf{inp}}$,
     (a) Parse $\mathsf{CT}_{\mathsf{tid},e} = (\mathsf{FE.ct}, \mathsf{FPFE.ct}_e)$.
     (b) $(\widehat{F}_{\mathsf{tid},v}, \mathsf{FPFE.sk}_{H_{\mathsf{tid},v}}) = \mathsf{FE.Dec}(\mathsf{FE.sk}_{G_{\mathsf{tid},v}}, \mathsf{FE.ct})$.
     (c) $\widehat{X}_{\mathsf{tid},e} = \mathsf{FPFE.Dec}(\mathsf{FPFE.sk}_{H_{\mathsf{tid},v}}, \mathsf{FPFE.ct}_e)$.
  3. For $e \in S_{\mathsf{in}}^{\mathsf{int}}$,
     (a) Parse $\mathsf{CT}_{\mathsf{tid},e} = (\mathsf{FE.ct}, \widehat{X}_{\mathsf{tid},e})$.
     (b) $(\widehat{F}_{\mathsf{tid},v}, \mathsf{FPFE.sk}_{H_{\mathsf{tid},v}}) = \mathsf{FE.Dec}(\mathsf{FE.sk}_{G_{\mathsf{tid},v}}, \mathsf{FE.ct})$.
  4. $(\{x_{\mathsf{tid},\alpha}\}_{\alpha\in S_{\mathsf{out}}^{\mathsf{out}}}, \{\widehat{X}_{\mathsf{tid},\beta}\}_{\beta\in S_{\mathsf{out}}}) = \mathsf{MHGC.Eval}(\widehat{F}_{\mathsf{tid},v}, \{\widehat{X}_{\mathsf{tid},e}\}_{e\in S_{\mathsf{in}}})$.
  5. For $\beta \in S_{\mathsf{out}}^{\mathsf{int}}$, $\mathsf{CT}_{\mathsf{tid},\beta} = (\mathsf{FE.ct}, \widehat{X}_{\mathsf{tid},\beta})$.

6. Output $(\{x_{\mathsf{tid},\alpha}\}_{\alpha\in S^{\mathsf{out}}_{\mathsf{out}}}, \{\mathsf{CT}_{\mathsf{tid},\beta}\}_{\beta\in S_{\mathsf{out}}})$.

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

$H^*[\mathsf{tid}, v, S_{\mathsf{in}}, \{c_{\mathsf{tid},e}\}_{e\in S_{\mathsf{in}}}](x_e, e, \mathsf{branch}_2, \mathsf{SKE}_2.\mathsf{sk})$:

1. If $e \notin S_{\mathsf{in}}$, output $\perp$.
2. Output $\widehat{X}_{\mathsf{tid},e} \leftarrow \mathsf{SKE}_2.\mathsf{Dec}(\mathsf{SKE}_2.\mathsf{sk}, c_{\mathsf{tid},e})$.

Figure 7: Definition of $H^*_{\mathsf{tid},v}$

6.3 Correctness

Efficiency: Using our discussion above on parameters, it is easy to see that the size and runtime of all algorithms of our MHGC scheme on security parameter 1^λ, maximum degree d_{max}, function size $\ell_{\mathcal{F}}$, message size $\ell_{\mathcal{X}}$, and type function type are $\mathsf{poly}(\lambda, d_{\mathsf{max}}, \ell_{\mathcal{F}}, \ell_{\mathcal{X}}, |\mathsf{type}|)$.

Correctness: We can prove correctness by considering a topological ordering on the nodes of the induced graph formed by the message and function queries, and proving inductively that MHFE.Dec outputs the correct values at each node.

Lemma 6.3 (Correctness of the bootstrapping construction MHGC $\Rightarrow$ MHFE)**.** *Assume that the underlying schemes* FE*,* FPFE*, and* MHGC *satisfy their standard correctness guarantees. Then the* MHFE *scheme defined in the bootstrapping construction satisfies correctness in the sense of Definition 4.8.*

Proof. Fix any polynomially bounded parameters $(\lambda, d_{\mathsf{max}}, \ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \mathsf{type})$, a topology identifier tid, and a valid query set

$$W = \big(\{(x_i, e_i)\}_{i\in[n]}, \{(\mathsf{tid}, f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j\in[m]}\big).$$

Let $\pi = (G, \mathcal{F}, \mathcal{X}, \mathsf{code}, \mathsf{data})$ be the induced transcript of W (Definition 4.5), where vertices are evaluated in a topological order of G. Let $\pi' = (G, \mathcal{F}', \mathcal{X}', \mathsf{code}', \mathsf{data}')$ be the induced transcript for the modified query set W' from Definition 4.8, where encryptions and keys are generated by the honest algorithms MHFE.Setup, MHFE.EncSetup, MHFE.Enc, MHFE.KeyGen and each node v_j is evaluated by $\mathsf{MHFE.Dec}(\mathsf{SK}_{\mathsf{tid},v_j}, \cdot)$.

We prove correctness by induction over a topological ordering of the computation vertices in G, showing that whenever a node is evaluated, MHFE.Dec outputs exactly the values prescribed by the underlying distributed computation on the corresponding plaintexts.

After processing the first $t-1$ computation vertices in topological order, for every edge e whose value has been defined by π (i.e., $\mathsf{data}(e) \neq \perp$), the transcript π' holds: (i) for $e \in E^{\mathsf{inp}}$, an MHFE ciphertext $\mathsf{CT}_{\mathsf{tid},e} = (\mathsf{FE.ct}, \mathsf{FPFE.ct}_e)$ that encrypts $(x_e, e, \cdot)$ for the correct $x_e = \mathsf{data}(e)$; and (ii) for $e \in E^{\mathsf{int}}$, an MHFE ciphertext $\mathsf{CT}_{\mathsf{tid},e} = (\mathsf{FE.ct}, \widehat{X}_{\mathsf{tid},e})$ where $\widehat{X}_{\mathsf{tid},e}$ is a valid MHGC-encoding of the correct internal-wire value $x_e = \mathsf{data}(e)$ (under the MHGC instance associated with topology tid).

Initially, π sets $\mathsf{data}(e_i) = x_i$ for message edges $e_i \in E^{\mathsf{inp}}$. In π', each such edge carries $\mathsf{CT}_{\mathsf{tid},e_i} = (\mathsf{FE.ct}, \mathsf{FPFE.ct}_{e_i})$ created by MHFE.Enc, which indeed encrypts $(x_i, e_i, \cdot)$ under FPFE

and shares the common $\mathsf{FE.ct}$ from $\mathsf{MHFE.EncSetup}$. Thus the hypothesis holds for all defined edges prior to any function evaluation.

Consider the t-th computation vertex in topological order, say $v = v_j$ with function $f = f_j$, incoming edges $S_{\mathsf{in}} = S_{\mathsf{in},j}$ and outgoing edges $S_{\mathsf{out}} = S_{\mathsf{out},j}$.

If any incoming edge $e \in S_{\mathsf{in}}$ satisfies $\mathsf{data}(e) = \perp$, then by Definition 4.5 the plaintext transcript π does not define any outgoing value at this node (equivalently, f on $\perp$-inputs yields $\perp$-outputs). The same happens in π': since some incoming edge value is $\perp$, the corresponding evaluation step in the induced transcript for W' is skipped, and no outgoing edge becomes defined. The induction hypothesis remains true.

Otherwise, all incoming edges in S_{in} are defined in π, so π computes

$$f(\{x_e\}_{e \in S_{\mathsf{in}}}) = \{x_{e'}\}_{e' \in S_{\mathsf{out}}} \qquad \text{where } x_e = \mathsf{data}(e).$$

We show that $\mathsf{MHFE.Dec}$ outputs these same values and produces correctly formatted outgoing ciphertexts.

1. By construction, $\mathsf{SK}_{\mathsf{tid},v} = \mathsf{FE.sk}_{G_{\mathsf{tid},v}}$ and every incoming ciphertext contains the same $\mathsf{FE.ct}$ produced in $\mathsf{MHFE.EncSetup}$. Therefore, correctness of FE implies that

   $$(\widehat{F}_{\mathsf{tid},v}, \mathsf{FPFE.sk}_{H_{\mathsf{tid},v}}) = \mathsf{FE.Dec}(\mathsf{FE.sk}_{G_{\mathsf{tid},v}}, \mathsf{FE.ct}) = G_{\mathsf{tid},v}(\mathsf{FPFE.msk}, \mathsf{PRF}_1.K, \mathsf{PRF}_2.K, 0, 0^{\ell_{\mathsf{SKE}_1 \cdot k}}),$$

   namely, decryption yields the intended MHGC garbled circuit $\widehat{F}_{\mathsf{tid},v}$ for f at node v (with input set S_{in} and output set S_{out}) and the FPFE key $\mathsf{FPFE.sk}_{H_{\mathsf{tid},v}}$ that will be used to produce garbled inputs for the *input* wires of this node.

2. For each incoming *input* edge $e \in S_{\mathsf{in}}{}^{\mathsf{inp}}$, the ciphertext has the form $\mathsf{CT}_{\mathsf{tid},e} = (\mathsf{FE.ct}, \mathsf{FPFE.ct}_e)$ where $\mathsf{FPFE.ct}_e$ encrypts $(x_e, e, 0, 0^{\ell_{\mathsf{SKE}_2 \cdot k}})$. Hence, correctness of FPFE implies

   $$\widehat{X}_{\mathsf{tid},e} = \mathsf{FPFE.Dec}(\mathsf{FPFE.sk}_{H_{\mathsf{tid},v}}, \mathsf{FPFE.ct}_e) = H_{\mathsf{tid},v}(x_e, e, 0, 0^{\ell_{\mathsf{SKE}_2 \cdot k}}).$$

   That is, FPFE decryption yields the correct MHGC garbled-input encoding $\widehat{X}_{\mathsf{tid},e}$ corresponding to the plaintext value x_e on edge e.

3. For each incoming *internal* edge $e \in S_{\mathsf{in}}{}^{\mathsf{int}}$, the ciphertext has the form $\mathsf{CT}_{\mathsf{tid},e} = (\mathsf{FE.ct}, \widehat{X}_{\mathsf{tid},e})$. By the induction hypothesis, $\widehat{X}_{\mathsf{tid},e}$ is already a valid MHGC encoding of the correct internal-wire value $x_e = \mathsf{data}(e)$.

4. Now $\mathsf{MHFE.Dec}$ invokes

   $$(\{x_{\mathsf{tid},\alpha}\}_{\alpha \in S_{\mathsf{out}}{}^{\mathsf{out}}}, \{\widehat{X}_{\mathsf{tid},\beta}\}_{\beta \in S_{\mathsf{out}}{}^{\mathsf{int}}}) = \mathsf{MHGC.Eval}(\widehat{F}_{\mathsf{tid},v}, \{\widehat{X}_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}}).$$

   By correctness of MHGC, since each $\widehat{X}_{\mathsf{tid},e}$ encodes the correct x_e on the incoming wires, the outputs satisfy:

   - for every output edge $\alpha \in S_{\mathsf{out}}{}^{\mathsf{out}}$, the plaintext value $x_{\mathsf{tid},\alpha}$ equals the correct distributed-computation value $x_\alpha = \mathsf{data}(\alpha)$ produced by applying f to the incoming plaintexts; and
   - for every internal edge $\beta \in S_{\mathsf{out}}{}^{\mathsf{int}}$, the string $\widehat{X}_{\mathsf{tid},\beta}$ is a valid MHGC encoding of the correct internal-wire value $x_\beta = \mathsf{data}(\beta)$.

5. Finally, $\mathsf{MHFE.Dec}$ outputs the clear values $\{x_{\mathsf{tid},\alpha}\}_{\alpha \in S_{\mathsf{out}}{}^{\mathsf{out}}}$, and for each internal $\beta \in S_{\mathsf{out}}{}^{\mathsf{int}}$ it sets

$$\mathsf{CT}_{\mathsf{tid},\beta} := (\mathsf{FE.ct}, \widehat{X}_{\mathsf{tid},\beta}).$$

Thus, the outgoing values written by π' match exactly those written by π on output edges, and the produced internal-edge ciphertexts satisfy the induction hypothesis for downstream nodes.

This completes the inductive step.

Thus by induction over the topological order, for every output edge $e \in E^{\mathsf{out}}$ that becomes defined in the plaintext induced transcript π, the value produced in π' equals the same plaintext value except with negligible probability (accumulating over polynomially many uses of $\mathsf{FE}, \mathsf{FPFE}, \mathsf{MHGC}$ correctness). Therefore, MHFE satisfies Definition 4.8. □

6.4 Security

We now formally prove Theorem 6.1 via a hybrid argument.

Remark 6.4. For all hybrids, we will assume when running the hybrid, we immediately halt and output 0 if the adversary ever aborts or if for any tid queried by the adversary,

$$W = (\{(x_i, e_i)\}_{i\in[n]}, \{(\mathsf{tid}, f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j\in[m]})$$

is ever **not** a valid query set with parameters $(\lambda, d_{\mathsf{max}}, \ell_{\mathcal{F}}, \ell_{\mathcal{X}}, \mathsf{type})$ queried in **topological order**, where $\{(x_i, e_i)\}_{i\in[n]}$ are all challenge message queries made by $\mathcal{A}$ and $\{(\mathsf{tid}, f_j, v_j, S_{\mathsf{in},j}, S_{\mathsf{out},j})\}_{j\in[m]}$ are the function queries made by $\mathcal{A}$ to topology id tid.

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

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs parameters $(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.
2. **Setup:**
   (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
   (b) Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.
3. **Encryption Setup:**
   (a) $\mathsf{PRF}_1.K \leftarrow \mathsf{PRF}_1.\mathsf{Setup}(1^\lambda)$.
   (b) $\mathsf{PRF}_2.K \leftarrow \mathsf{PRF}_2.\mathsf{Setup}(1^\lambda)$.
   (c) $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$.
   (d) $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (\mathsf{FPFE.msk}, \mathsf{PRF}_1.K, \mathsf{PRF}_2.K, 0, 0^{\ell_{\mathsf{SKE}_1.\mathsf{sk}}}))$.
4. The adversary can make a polynomial number of function and message queries in topological order.

- **Function Query:** On function query $(\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$:
  1. Sample $c_{\mathsf{tid},v} \leftarrow \{0,1\}^{\ell_{\mathsf{SKE}_1.\mathsf{ct}}}$.
  2. For $e \in S_{\mathsf{in}}$, sample $c_{\mathsf{tid},e} \leftarrow \{0,1\}^{\ell_{\mathsf{SKE}_2.\mathsf{ct}}}$.
  3. Let $G_{\mathsf{tid},v} = G[\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, c_{\mathsf{tid},v}, \{c_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}}]$ as defined in Figure 5.
  4. $\mathsf{FE.sk}_{G_{\mathsf{tid},v}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_{\mathsf{tid},v})$.
  5. Send $\mathsf{SK}_{\mathsf{tid},v} = \mathsf{FE.sk}_{G_{\mathsf{tid},v}}$ to the adversary.
- **Message Query:** On message query (x_e, e):
  1. $\mathsf{FPFE.ct}_e \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (x_e, e, 0, 0^{\ell_{\mathsf{SKE}_2.k}}))$.
  2. $\mathsf{CT}_e = (\mathsf{FE.ct}, \mathsf{FPFE.ct}_e)$.
  3. Send CT_e to the adversary.

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

Hybrid$_1^{\mathcal{A}}(1^\lambda)$: For each node, we hardwire into each $c_{\mathsf{tid},v}$ the values that would be output by G on the challenge queries.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs parameters $(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.
2. **Setup:**
   (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
   (b) Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.
3. **Encryption Setup:**
   (a) $\mathsf{PRF}_1.K \leftarrow \mathsf{PRF}_1.\mathsf{Setup}(1^\lambda)$.
   (b) $\mathsf{PRF}_2.K \leftarrow \mathsf{PRF}_2.\mathsf{Setup}(1^\lambda)$.
   (c) $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$.
   (d) $\mathsf{SKE}_1.\mathsf{sk} \leftarrow \mathsf{SKE}_1.\mathsf{Setup}(1^\lambda)$.
   (e) $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (\mathsf{FPFE.msk}, \mathsf{PRF}_1.K, \mathsf{PRF}_2.K, 0, 0^{\ell_{\mathsf{SKE}_1.\mathsf{sk}}}))$.
4. The adversary can make a polynomial number of function and message queries in topological order.

- **Function Query:** On function query $(\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$:
  1. ~~Sample $c_{\mathsf{tid},v} \leftarrow \{0,1\}^{\ell_{\mathsf{SKE.ct}}}$.~~
  2. ~~For $e \in S_{\mathsf{in}}$, sample $c_{\mathsf{tid},e} \leftarrow \{0,1\}^{\ell_{\mathsf{SKE.ct}}}$.~~
  3. **Compute Hardwired Values:**
     (a) If this is the first query to tid, compute the following:
        i. $(r_{\mathsf{Setup}}, r_{\mathsf{PRF}}) \leftarrow \mathsf{PRF}_1.\mathsf{Eval}(\mathsf{PRF}_1.K, \mathsf{tid})$.
        ii. $\mathsf{MHGC.msk}_{\mathsf{tid}} \leftarrow \mathsf{MHGC.Setup}(1^\lambda; r_{\mathsf{Setup}})$.
        iii. $\mathsf{PRF}_3.K_{\mathsf{tid}} \leftarrow \mathsf{PRF}_3.\mathsf{Setup}(1^\lambda; r_{\mathsf{PRF}})$.
     (b) For $e \in S_{\mathsf{in}}$, sample $c_{\mathsf{tid},e} \leftarrow \{0,1\}^{\ell_{\mathsf{SKE.ct}}}$.
     (c) $(r_H, r_{\mathsf{GarbCirc}}) \leftarrow \mathsf{PRF}_2.\mathsf{Eval}(\mathsf{PRF}_2.K, (\mathsf{tid}, v))$.
     (d) Let $H_{\mathsf{tid},v} = H[\mathsf{tid}, v, S_{\mathsf{in}}, \mathsf{MHGC.msk}_{\mathsf{tid}}, \mathsf{PRF}_3.K_{\mathsf{tid}}, \{c_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}}]$ as defined in Figure 6.
     (e) $\mathsf{FPFE.sk}_{H_{\mathsf{tid},v}} \leftarrow \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H_{\mathsf{tid},v}; r_H)$.
     (f) $\widehat{F}_{\mathsf{tid},v} \leftarrow \mathsf{MHGC.GarbCirc}(\mathsf{MHGC.msk}_{\mathsf{tid}}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}}; r_{\mathsf{GarbCirc}})$.
     (g) $c_{\mathsf{tid},v} \leftarrow \mathsf{SKE}_1.\mathsf{Enc}(\mathsf{SKE}_1.\mathsf{sk}, (\widehat{F}_{\mathsf{tid},v}, \mathsf{FPFE.sk}_{H_{\mathsf{tid},v}}))$.
  4. Let $G_{\mathsf{tid},v} = G[\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, c_{\mathsf{tid},v}, \{c_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}}]$ as defined in Figure 5.
  5. $\mathsf{FE.sk}_{G_{\mathsf{tid},v}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_{\mathsf{tid},v})$.
  6. Send $\mathsf{SK}_{\mathsf{tid},v} = \mathsf{FE.sk}_{G_{\mathsf{tid},v}}$ to the adversary.
- **Message Query:** On message query (x_e, e):
  1. $\mathsf{FPFE.ct}_e \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (x_e, e, 0, 0^{\ell_{\mathsf{SKE}_2 \cdot k}}))$.
  2. $\mathsf{CT}_e = (\mathsf{FE.ct}, \mathsf{FPFE.ct}_e)$.

3. Send CT_e to the adversary.

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

Lemma 6.5. *If* SKE_1 *is a secret key encryption scheme with pseudorandom ciphertexts, then for all PPT adversaries* $\mathcal{A}$*,*

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

Proof. We are replacing random values $c_{\mathsf{tid},v}$ with encrypted messages under SKE_1. Thus, the lemma follows by using a straightforward reduction to the pseudorandom ciphertext property of SKE_1. □

Hybrid$_2^{\mathcal{A}}(1^\lambda)$: We change the FE ciphertext to use the $\mathsf{branch}_1 = 1$ branch of each $G_{\mathsf{tid},v}$.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs parameters $(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.
2. **Setup:**
   (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
   (b) Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.
3. **Encryption Setup:**
   (a) $\mathsf{PRF}_1.K \leftarrow \mathsf{PRF}_1.\mathsf{Setup}(1^\lambda)$.
   (b) $\mathsf{PRF}_2.K \leftarrow \mathsf{PRF}_2.\mathsf{Setup}(1^\lambda)$.
   (c) $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$.
   (d) $\mathsf{SKE}_1.\mathsf{sk} \leftarrow \mathsf{SKE}_1.\mathsf{Setup}(1^\lambda)$.
   (e) $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (0^{\ell_{\mathsf{FPFE.msk}}}, 0^{\ell_{\mathsf{PRF}_1.K}}, 0^{\ell_{\mathsf{PRF}_2.K}}, 1, \mathsf{SKE}_1.\mathsf{sk}))$.
4. The adversary can make a polynomial number of function and message queries in topological order.

- **Function Query:** On function query $(\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$:
  1. **Compute Hardwired Values:**
     (a) If this is the first query to tid, compute the following:
        i. $(r_{\mathsf{Setup}}, r_{\mathsf{PRF}}) \leftarrow \mathsf{PRF}_1.\mathsf{Eval}(\mathsf{PRF}_1.K, \mathsf{tid})$.
        ii. $\mathsf{MHGC.msk}_{\mathsf{tid}} \leftarrow \mathsf{MHGC.Setup}(1^\lambda; r_{\mathsf{Setup}})$.
        iii. $\mathsf{PRF}_3.K_{\mathsf{tid}} \leftarrow \mathsf{PRF}_3.\mathsf{Setup}(1^\lambda; r_{\mathsf{PRF}})$.
     (b) For $e \in S_{\mathsf{in}}$, sample $c_{\mathsf{tid},e} \leftarrow \{0,1\}^{\ell_{\mathsf{SKE.ct}}}$.
     (c) $(r_H, r_{\mathsf{GarbCirc}}) \leftarrow \mathsf{PRF}_2.\mathsf{Eval}(\mathsf{PRF}_2.K, (\mathsf{tid}, v))$.
     (d) Let $H_{\mathsf{tid},v} = H[\mathsf{tid}, v, S_{\mathsf{in}}, \mathsf{MHGC.msk}_{\mathsf{tid}}, \mathsf{PRF}_3.K_{\mathsf{tid}}, \{c_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}}]$ as defined in Figure 6.
     (e) $\mathsf{FPFE.sk}_{H_{\mathsf{tid},v}} \leftarrow \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H_{\mathsf{tid},v}; r_H)$.
     (f) $\widehat{F}_{\mathsf{tid},v} \leftarrow \mathsf{MHGC.GarbCirc}(\mathsf{MHGC.msk}_{\mathsf{tid}}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}}; r_{\mathsf{GarbCirc}})$.
     (g) $c_{\mathsf{tid},v} \leftarrow \mathsf{SKE}_1.\mathsf{Enc}(\mathsf{SKE}_1.\mathsf{sk}, (\widehat{F}_{\mathsf{tid},v}, \mathsf{FPFE.sk}_{H_{\mathsf{tid},v}}))$.
  2. Let $G_{\mathsf{tid},v} = G[\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, c_{\mathsf{tid},v}, \{c_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}}]$ as defined in Figure 5.
  3. $\mathsf{FE.sk}_{G_{\mathsf{tid},v}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_{\mathsf{tid},v})$.
  4. Send $\mathsf{SK}_{\mathsf{tid},v} = \mathsf{FE.sk}_{G_{\mathsf{tid},v}}$ to the adversary.
- **Message Query:** On message query (x_e, e):
  1. $\mathsf{FPFE.ct}_e \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (x_e, e, 0, 0^{\ell_{\mathsf{SKE}_2 \cdot k}}))$.
  2. $\mathsf{CT}_e = (\mathsf{FE.ct}, \mathsf{FPFE.ct}_e)$.
  3. Send CT_e to the adversary.
5. **Experiment Outcome**: $\mathcal{A}$ outputs a bit b' which is the output of the experiment.

Lemma 6.6. *If* FE *is an adaptive-IND-secure functional encryption scheme, then for all PPT adversaries* $\mathcal{A}$*,*

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

Proof. For all functions $G_{\mathsf{tid},v}$, we have

$$\begin{aligned}
&G[\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, c_{\mathsf{tid},v}, \{c_{\mathsf{tid},e}\}_{e\in S_{\mathsf{in}}}](0^{\ell_{\mathsf{FPFE.msk}}}, 0^{\ell_{\mathsf{PRF}_1.K}}, 0^{\ell_{\mathsf{PRF}_2.K}}, 1, \mathsf{SKE}_1.\mathsf{sk}) \\
&= G[\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, c_{\mathsf{tid},v}, \{c_{\mathsf{tid},e}\}_{e\in S_{\mathsf{in}}}](\mathsf{FPFE.msk}, \mathsf{PRF}_1.K, \mathsf{PRF}_2.K, 0, 0^{\ell_{\mathsf{SKE}_1.\mathsf{sk}}})
\end{aligned}$$

since we previously set each ciphertext $c_{\mathsf{tid},v}$ to contain the original output value of each $G_{\mathsf{tid},v}$. Thus, we can use the adaptive security of the FE scheme to swap the FE ciphertext $\mathsf{FE.ct}$ from an encryption of $(\mathsf{FPFE.msk}, \mathsf{PRF}_1.K, \mathsf{PRF}_2.K, 0, 0^{\ell_{\mathsf{SKE}_1.\mathsf{sk}}})$ to an an encryption of $(0^{\ell_{\mathsf{FPFE.msk}}}, 0^{\ell_{\mathsf{PRF}_1.K}}, 0^{\ell_{\mathsf{PRF}_2.K}}, 1, \mathsf{SKE}_1.\mathsf{sk})$. □

Hybrid$_3^{\mathcal{A}}(1^\lambda)$: We change the randomness produced by PRF_1 and PRF_2 to true randomness.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs parameters $(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.
2. **Setup:**
   (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
   (b) Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.
3. **Encryption Setup:**
   (a) ~~$\mathsf{PRF}_1.K \leftarrow \mathsf{PRF}_1.\mathsf{Setup}(1^\lambda)$.~~
   (b) ~~$\mathsf{PRF}_2.K \leftarrow \mathsf{PRF}_2.\mathsf{Setup}(1^\lambda)$.~~
   (c) $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$.
   (d) $\mathsf{SKE}_1.\mathsf{sk} \leftarrow \mathsf{SKE}_1.\mathsf{Setup}(1^\lambda)$.
   (e) $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (0^{\ell_{\mathsf{FPFE.msk}}}, 0^{\ell_{\mathsf{PRF}_1.K}}, 0^{\ell_{\mathsf{PRF}_2.K}}, 1, \mathsf{SKE}_1.\mathsf{sk}))$.
4. The adversary can make a polynomial number of function and message queries in topological order.

- **Function Query:** On function query $(\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$:
  1. **Compute Hardwired Values:**
     (a) If this is the first query to tid, compute the following:
        i. ~~$(r_{\mathsf{Setup}}, r_{\mathsf{PRF}}) \leftarrow \mathsf{PRF}_1.\mathsf{Eval}(\mathsf{PRF}_1.K, \mathsf{tid})$.~~
        ii. $\mathsf{MHGC.msk}_{\mathsf{tid}} \leftarrow \mathsf{MHGC.Setup}(1^\lambda; r_{\mathsf{Setup}})$.
        iii. $\mathsf{PRF}_3.K_{\mathsf{tid}} \leftarrow \mathsf{PRF}_3.\mathsf{Setup}(1^\lambda; r_{\mathsf{PRF}})$.
     (b) For $e \in S_{\mathsf{in}}$, sample $c_{\mathsf{tid},e} \leftarrow \{0,1\}^{\ell_{\mathsf{SKE.ct}}}$.
     (c) $(r_H, r_{\mathsf{GarbCirc}}) \leftarrow \mathsf{PRF}_2.\mathsf{Eval}(\mathsf{PRF}_2.K, (\mathsf{tid}, v))$.
     (d) Let $H_{\mathsf{tid},v} = H[\mathsf{tid}, v, S_{\mathsf{in}}, \mathsf{MHGC.msk}_{\mathsf{tid}}, \mathsf{PRF}_3.K_{\mathsf{tid}}, \{c_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}}]$ as defined in Figure 6.
     (e) $\mathsf{FPFE.sk}_{H_{\mathsf{tid},v}} \leftarrow \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H_{\mathsf{tid},v}; r_H)$.
     (f) $\widehat{F}_{\mathsf{tid},v} \leftarrow \mathsf{MHGC.GarbCirc}(\mathsf{MHGC.msk}_{\mathsf{tid}}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}}; r_{\mathsf{GarbCirc}})$.
     (g) $c_{\mathsf{tid},v} \leftarrow \mathsf{SKE}_1.\mathsf{Enc}(\mathsf{SKE}_1.\mathsf{sk}, (\widehat{F}_{\mathsf{tid},v}, \mathsf{FPFE.sk}_{H_{\mathsf{tid},v}}))$.
  2. Let $G_{\mathsf{tid},v} = G[\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, c_{\mathsf{tid},v}, \{c_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}}]$ as defined in Figure 5.
  3. $\mathsf{FE.sk}_{G_{\mathsf{tid},v}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_{\mathsf{tid},v})$.
  4. Send $\mathsf{SK}_{\mathsf{tid},v} = \mathsf{FE.sk}_{G_{\mathsf{tid},v}}$ to the adversary.
- **Message Query:** On message query (x_e, e):
  1. $\mathsf{FPFE.ct}_e \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (x_e, e, 0, 0^{\ell_{\mathsf{SKE}_2 \cdot k}}))$.
  2. $\mathsf{CT}_e = (\mathsf{FE.ct}, \mathsf{FPFE.ct}_e)$.
  3. Send CT_e to the adversary.
5. **Experiment Outcome**: $\mathcal{A}$ outputs a bit b' which is the output of the experiment.

Lemma 6.7. *If* PRF_1 *and* PRF_2 *are secure* PRF*s, then for all PPT adversaries* $\mathcal{A}$*,*

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

Proof. Note that the $\mathsf{PRF}_1.K$ and $\mathsf{PRF}_2.K$ are no longer encrypted inside $\mathsf{FE.ct}$. Thus, this follows by a straightforward reduction to the security of the PRF schemes. □

Hybrid$_4^{\mathcal{A}}(1^\lambda)$: For each function query and each input edge $e \in S_{\mathsf{in}}$, we hardwire into $c_{\mathsf{tid},e}$ the value that would be output by $H_{\mathsf{tid},v}$ on the challenge message query for edge e (if it exists). Since the queries must be given in topological order, we will know all input messages to a function, before we are asked for a function key for that node.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs parameters $(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.
2. **Setup:**
   (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
   (b) Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.
3. **Encryption Setup:**
   (a) $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$.
   (b) $\mathsf{SKE}_1.\mathsf{sk} \leftarrow \mathsf{SKE}_1.\mathsf{Setup}(1^\lambda)$.
   (c) $\mathsf{SKE}_2.\mathsf{sk} \leftarrow \mathsf{SKE}_2.\mathsf{Setup}(1^\lambda)$.
   (d) $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (0^{\ell_{\mathsf{FPFE.msk}}}, 0^{\ell_{\mathsf{PRF}_1.K}}, 0^{\ell_{\mathsf{PRF}_2.K}}, 1, \mathsf{SKE}_1.\mathsf{sk}))$.
4. The adversary can make a polynomial number of function and message queries in topological order.

- **Function Query:** On function query $(\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$:
  1. **Compute Hardwired Values:**
     (a) If this is the first query to tid, compute the following:
        i. $\mathsf{MHGC.msk}_{\mathsf{tid}} \leftarrow \mathsf{MHGC.Setup}(1^\lambda)$.
        ii. $\mathsf{PRF}_3.K_{\mathsf{tid}} \leftarrow \mathsf{PRF}_3.\mathsf{Setup}(1^\lambda)$.
     (b) ~~For $e \in S_{\mathsf{in}}$, sample $c_{\mathsf{tid},e} \leftarrow \{0,1\}^{\ell_{\mathsf{SKE.ct}}}$.~~
     (c) For $e \in S_{\mathsf{in}}$,
        i. If x_e is already defined (as a message query),
           A. $r_{\mathsf{GarbInp}} \leftarrow \mathsf{PRF}_3.\mathsf{Eval}(\mathsf{PRF}_3.K_{\mathsf{tid}}, e)$.
           B. $\widehat{X}_{\mathsf{tid},e} \leftarrow \mathsf{MHGC.GarbInp}(\mathsf{MHGC.msk}_{\mathsf{tid}}, x_e, e; r_{\mathsf{GarbInp}})$.
           C. $c_{\mathsf{tid},e} \leftarrow \mathsf{SKE}_2.\mathsf{Enc}(\mathsf{SKE}_2.\mathsf{sk}, \widehat{X}_{\mathsf{tid},e})$.
        ii. Else, $c_{\mathsf{tid},e} \leftarrow \{0,1\}^{\ell_{\mathsf{SKE}_2.\mathsf{ct}}}$.
     (d) Let $H_{\mathsf{tid},v} = H[\mathsf{tid}, v, S_{\mathsf{in}}, \mathsf{MHGC.msk}_{\mathsf{tid}}, \mathsf{PRF}_3.K_{\mathsf{tid}}, \{c_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}}]$ as defined in Figure 6.
     (e) $\mathsf{FPFE.sk}_{H_{\mathsf{tid},v}} \leftarrow \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H_{\mathsf{tid},v})$.
     (f) $\widehat{F}_{\mathsf{tid},v} \leftarrow \mathsf{MHGC.GarbCirc}(\mathsf{MHGC.msk}_{\mathsf{tid}}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$.
     (g) $c_{\mathsf{tid},v} \leftarrow \mathsf{SKE}_1.\mathsf{Enc}(\mathsf{SKE}_1.\mathsf{sk}, (\widehat{F}_{\mathsf{tid},v}, \mathsf{FPFE.sk}_{H_{\mathsf{tid},v}}))$.
  2. Let $G_{\mathsf{tid},v} = G[\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, c_{\mathsf{tid},v}, \{c_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}}]$ as defined in Figure 5.
  3. $\mathsf{FE.sk}_{G_{\mathsf{tid},v}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_{\mathsf{tid},v})$.
  4. Send $\mathsf{SK}_{\mathsf{tid},v} = \mathsf{FE.sk}_{G_{\mathsf{tid},v}}$ to the adversary.
- **Message Query:** On message query (x_e, e):

1. $\mathsf{FPFE.ct}_e \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (x_e, e, 0, 0^{\ell_{\mathsf{SKE}_2 \cdot k}}))$.
2. $\mathsf{CT}_e = (\mathsf{FE.ct}, \mathsf{FPFE.ct}_e)$.
3. Send CT_e to the adversary.

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

Lemma 6.8. *If* SKE_2 *is a secret key encryption scheme with pseudorandom ciphertexts, then for all PPT adversaries* $\mathcal{A}$*,*

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

Proof. We are replacing random values $c_{\mathsf{tid},e}$ with encrypted messages under SKE_2. Thus, the lemma follows by using a straightforward reduction to the pseudorandom ciphertext property of SKE_2. □

Hybrid$_5^{\mathcal{A}}(1^\lambda)$: We change the FPFE ciphertexts to use the $\mathsf{branch}_2 = 1$ branch of each $H_{\mathsf{tid},v}$.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs parameters $(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.
2. **Setup:**
   (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
   (b) Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.
3. **Encryption Setup:**
   (a) $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$.
   (b) $\mathsf{SKE}_1.\mathsf{sk} \leftarrow \mathsf{SKE}_1.\mathsf{Setup}(1^\lambda)$.
   (c) $\mathsf{SKE}_2.\mathsf{sk} \leftarrow \mathsf{SKE}_2.\mathsf{Setup}(1^\lambda)$.
   (d) $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (0^{\ell_{\mathsf{FPFE.msk}}}, 0^{\ell_{\mathsf{PRF}_1.K}}, 0^{\ell_{\mathsf{PRF}_2.K}}, 1, \mathsf{SKE}_1.\mathsf{sk}))$.
4. The adversary can make a polynomial number of function and message queries in topological order.

- **Function Query:** On function query $(\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$:
  1. **Compute Hardwired Values:**
     (a) If this is the first query to tid, compute the following:
        i. $\mathsf{MHGC.msk}_{\mathsf{tid}} \leftarrow \mathsf{MHGC.Setup}(1^\lambda)$.
        ii. $\mathsf{PRF}_3.K_{\mathsf{tid}} \leftarrow \mathsf{PRF}_3.\mathsf{Setup}(1^\lambda)$.
     (b) For $e \in S_{\mathsf{in}}$,
        i. If x_e is already defined (as a message query),
           A. $r_{\mathsf{GarbInp}} \leftarrow \mathsf{PRF}_3.\mathsf{Eval}(\mathsf{PRF}_3.K_{\mathsf{tid}}, e)$.
           B. $\widehat{X}_{\mathsf{tid},e} \leftarrow \mathsf{MHGC.GarbInp}(\mathsf{MHGC.msk}_{\mathsf{tid}}, x_e, e; r_{\mathsf{GarbInp}})$.
           C. $c_{\mathsf{tid},e} \leftarrow \mathsf{SKE}_2.\mathsf{Enc}(\mathsf{SKE}_2.\mathsf{sk}, \widehat{X}_{\mathsf{tid},e})$.
        ii. Else, $c_{\mathsf{tid},e} \leftarrow \{0,1\}^{\ell_{\mathsf{SKE}_2.\mathsf{ct}}}$.
     (c) Let $H_{\mathsf{tid},v} = H[\mathsf{tid}, v, S_{\mathsf{in}}, \mathsf{MHGC.msk}_{\mathsf{tid}}, \mathsf{PRF}_3.K_{\mathsf{tid}}, \{c_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}}]$ as defined in Figure 6.
     (d) $\mathsf{FPFE.sk}_{H_{\mathsf{tid},v}} \leftarrow \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H_{\mathsf{tid},v})$.
     (e) $\widehat{F}_{\mathsf{tid},v} \leftarrow \mathsf{MHGC.GarbCirc}(\mathsf{MHGC.msk}_{\mathsf{tid}}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$.
     (f) $c_{\mathsf{tid},v} \leftarrow \mathsf{SKE}_1.\mathsf{Enc}(\mathsf{SKE}_1.\mathsf{sk}, (\widehat{F}_{\mathsf{tid},v}, \mathsf{FPFE.sk}_{H_{\mathsf{tid},v}}))$.
  2. Let $G_{\mathsf{tid},v} = G[\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, c_{\mathsf{tid},v}, \{c_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}}]$ as defined in Figure 5.
  3. $\mathsf{FE.sk}_{G_{\mathsf{tid},v}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_{\mathsf{tid},v})$.
  4. Send $\mathsf{SK}_{\mathsf{tid},v} = \mathsf{FE.sk}_{G_{\mathsf{tid},v}}$ to the adversary.
- **Message Query:** On message query (x_e, e):
  1. $\mathsf{FPFE.ct}_e \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (0^{\ell_{\mathcal{X}}}, e, 1, \mathsf{SKE}_2.\mathsf{sk}))$.
  2. $\mathsf{CT}_e = (\mathsf{FE.ct}, \mathsf{FPFE.ct}_e)$.
  3. Send CT_e to the adversary.

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

Lemma 6.9. *If* FPFE *is an adaptive-IND-secure functional encryption scheme, then for all PPT adversaries* $\mathcal{A}$*,*

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

Proof. For all functions $H_{\mathsf{tid},v}$ produced and all message queries (x_e, e), we have

$$\begin{aligned}
&H[\mathsf{tid}, v, S_{\mathsf{in}}, \mathsf{MHGC.msk}_{\mathsf{tid}}, \mathsf{PRF}_3.K_{\mathsf{tid}}, \{c_{\mathsf{tid},e}\}_{e\in S_{\mathsf{in}}}](0^{\ell_{\mathcal{X}}}, e, 1, \mathsf{SKE}_2.\mathsf{sk}))\\
&= H[\mathsf{tid}, v, S_{\mathsf{in}}, \mathsf{MHGC.msk}_{\mathsf{tid}}, \mathsf{PRF}_3.K_{\mathsf{tid}}, \{c_{\mathsf{tid},e}\}_{e\in S_{\mathsf{in}}}](x_e, e, 0, 0^{\ell_{\mathsf{SKE.sk}}}))
\end{aligned}$$

since we previously set each ciphertext $c_{\mathsf{tid},ev}$ to contain the original output value of each $H_{\mathsf{tid},v}$. Thus, we can use the security of the FPFE scheme to swap the FPFE ciphertexts $\mathsf{FPFE.ct}_e$ from encryptions of $(x_e, e, 0, 0^{\ell_{\mathsf{SKE.sk}}})$ to encryptions of $(0^{\ell_{\mathcal{X}}}, e, 1, \mathsf{SKE}_2.\mathsf{sk})$. Note that the FPFE master secret key is no longer in $\mathsf{FE.ct}$ and that the restriction on a topological ordering of queries requires any input messsage queries to a node to be queried before that node. □

Hybrid$_6^{\mathcal{A}}(1^\lambda)$: We use the function privacy of FPFE to change from creating function keys for $H_{\mathsf{tid},v}$ to function keys for $H^*_{\mathsf{tid},v}$.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs parameters $(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.
2. **Setup:**
   (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
   (b) Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.
3. **Encryption Setup:**
   (a) $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$.
   (b) $\mathsf{SKE}_1.\mathsf{sk} \leftarrow \mathsf{SKE}_1.\mathsf{Setup}(1^\lambda)$.
   (c) $\mathsf{SKE}_2.\mathsf{sk} \leftarrow \mathsf{SKE}_2.\mathsf{Setup}(1^\lambda)$.
   (d) $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (0^{\ell_{\mathsf{FPFE.msk}}}, 0^{\ell_{\mathsf{PRF}_1 \cdot K}}, 0^{\ell_{\mathsf{PRF}_2 \cdot K}}, 1, \mathsf{SKE}_1.\mathsf{sk}))$.
4. The adversary can make a polynomial number of function and message queries in topological order.

- **Function Query:** On function query $(\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$:
  1. **Compute Hardwired Values:**
     (a) If this is the first query to tid, compute the following:
        i. $\mathsf{MHGC.msk}_{\mathsf{tid}} \leftarrow \mathsf{MHGC.Setup}(1^\lambda)$.
        ii. $\mathsf{PRF}_3.K_{\mathsf{tid}} \leftarrow \mathsf{PRF}_3.\mathsf{Setup}(1^\lambda)$.
     (b) For $e \in S_{\mathsf{in}}$,
        i. If x_e is already defined (as a message query),
           A. $r_{\mathsf{GarbInp}} \leftarrow \mathsf{PRF}_3.\mathsf{Eval}(\mathsf{PRF}_3.K_{\mathsf{tid}}, e)$.
           B. $\widehat{X}_{\mathsf{tid},e} \leftarrow \mathsf{MHGC.GarbInp}(\mathsf{MHGC}_{\mathsf{tid}}.\mathsf{msk}, x_e, e; r_{\mathsf{GarbInp}})$.
           C. $c_{\mathsf{tid},e} \leftarrow \mathsf{SKE}_2.\mathsf{Enc}(\mathsf{SKE}_2.\mathsf{sk}, \widehat{X}_{\mathsf{tid},e})$.
        ii. Else, $c_{\mathsf{tid},e} \leftarrow \{0,1\}^{\ell_{\mathsf{SKE}_2.\mathsf{ct}}}$.
     (c) Let $H_{\mathsf{tid},v} = H^*[\mathsf{tid}, v, S_{\mathsf{in}}, \{c_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}}]$ as defined in Figure 7.
     (d) $\mathsf{FPFE.sk}_{H_{\mathsf{tid},v}} \leftarrow \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H_{\mathsf{tid},v})$.
     (e) $\widehat{F}_{\mathsf{tid},v} \leftarrow \mathsf{MHGC.GarbCirc}(\mathsf{MHGC.msk}_{\mathsf{tid}}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$.
     (f) $c_{\mathsf{tid},v} \leftarrow \mathsf{SKE}_1.\mathsf{Enc}(\mathsf{SKE}_1.\mathsf{sk}, (\widehat{F}_{\mathsf{tid},v}, \mathsf{FPFE.sk}_{H_{\mathsf{tid},v}}))$.
  2. Let $G_{\mathsf{tid},v} = G[\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, c_{\mathsf{tid},v}, \{c_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}}]$ as defined in Figure 5.
  3. $\mathsf{FE.sk}_{G_{\mathsf{tid},v}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_{\mathsf{tid},v})$.
  4. Send $\mathsf{SK}_{\mathsf{tid},v} = \mathsf{FE.sk}_{G_{\mathsf{tid},v}}$ to the adversary.
- **Message Query:** On message query (x_e, e):
  1. $\mathsf{FPFE.ct}_e \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (0^{\ell_{\mathcal{X}}}, e, 1, \mathsf{SKE}_2.\mathsf{sk}))$.
  2. $\mathsf{CT}_e = (\mathsf{FE.ct}, \mathsf{FPFE.ct}_e)$.
  3. Send CT_e to the adversary.

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

Lemma 6.10. *If* FPFE *is a function-private FE scheme, then for all PPT adversaries* $\mathcal{A}$*,*

$$\left|\Pr[\mathbf{Hybrid}_5^{\mathcal{A}}(1^\lambda)] - \Pr[\mathbf{Hybrid}_6^{\mathcal{A}}(1^\lambda)]\right| \leq \mathsf{negl}(\lambda).$$

Proof. For all function queries to (tid, v) and all message queries for (x_e, e), we have

$$\begin{aligned}
&H[\mathsf{tid}, v, S_{\mathsf{in}}, \mathsf{MHFE.msk}_{\mathsf{tid}}, \mathsf{PRF}_3.K_{\mathsf{tid}}, \{c_{\mathsf{tid},e}\}_{e\in S_{\mathsf{in}}}](0^{\ell_{\mathcal{X}}}, e, 1, \mathsf{SKE}_2.\mathsf{sk}))\\
&= H^*[\mathsf{tid}, v, S_{\mathsf{in}}, \{c_{\mathsf{tid},e}\}_{e\in S_{\mathsf{in}}}](0^{\ell_{\mathcal{X}}}, e, 1, \mathsf{SKE}_2.\mathsf{sk}))
\end{aligned}$$

since we are no longer using the $\mathsf{branch}_2 = 0$ branch of $H_{\mathsf{tid},v}$. Thus, we can use the function privacy security of the FPFE scheme to swap the FPFE function keys from keys $\mathsf{FPFE.sk}_{H_{\mathsf{tid},v}}$ for $H^*_{\mathsf{tid},v}$ to keys $\mathsf{FPFE.sk}_{H^*_{\mathsf{tid},v}}$ for $H^*_{\mathsf{tid},v}$. Note that the FPFE master secret key is no longer in $\mathsf{FE.ct}$. □

Hybrid$_7^{\mathcal{A}}(1^\lambda)$: We change the values output by PRF_3 to random values.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs parameters $(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.
2. **Setup:**
   (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
   (b) Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.
3. **Encryption Setup:**
   (a) $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$.
   (b) $\mathsf{SKE}_1.\mathsf{sk} \leftarrow \mathsf{SKE}_1.\mathsf{Setup}(1^\lambda)$.
   (c) $\mathsf{SKE}_2.\mathsf{sk} \leftarrow \mathsf{SKE}_2.\mathsf{Setup}(1^\lambda)$.
   (d) $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (0^{\ell_{\mathsf{FPFE.msk}}}, 0^{\ell_{\mathsf{PRF}_1 \cdot K}}, 0^{\ell_{\mathsf{PRF}_2 \cdot K}}, 1, \mathsf{SKE}_1.\mathsf{sk}))$.
4. The adversary can make a polynomial number of function and message queries in topological order.

- **Function Query:** On function query $(\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$:
  1. **Compute Hardwired Values:**
     (a) If this is the first query to tid, compute the following:
        i. $\mathsf{MHGC.msk}_{\mathsf{tid}} \leftarrow \mathsf{MHGC.Setup}(1^\lambda)$.
        ii. ~~$\mathsf{PRF}_3.K_{\mathsf{tid}} \leftarrow \mathsf{PRF}_3.\mathsf{Setup}(1^\lambda)$.~~
     (b) For $e \in S_{\mathsf{in}}$,
        i. If x_e is already defined (as a message query),
           A. ~~$r_{\mathsf{GarbInp}} \leftarrow \mathsf{PRF}_3.\mathsf{Eval}(\mathsf{PRF}_3.K_{\mathsf{tid}}, e)$.~~
           B. $\widehat{X}_{\mathsf{tid},e} \leftarrow \mathsf{MHGC.GarbInp}(\mathsf{MHGC.msk}_{\mathsf{tid}}, x_e, e; ~~r_{\mathsf{GarbInp}}~~)$.
           C. $c_{\mathsf{tid},e} \leftarrow \mathsf{SKE}_2.\mathsf{Enc}(\mathsf{SKE}_2.\mathsf{sk}, \widehat{X}_{\mathsf{tid},e})$.
        ii. Else, $c_{\mathsf{tid},e} \leftarrow \{0,1\}^{\ell_{\mathsf{SKE}_2.\mathsf{ct}}}$.
     (c) Let $H_{\mathsf{tid},v} = H^*[\mathsf{tid}, v, S_{\mathsf{in}}, \{c_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}}]$ as defined in Figure 7.
     (d) $\mathsf{FPFE.sk}_{H_{\mathsf{tid},v}} \leftarrow \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H_{\mathsf{tid},v})$.
     (e) $\widehat{F}_{\mathsf{tid},v} \leftarrow \mathsf{MHGC.GarbCirc}(\mathsf{MHGC.msk}_{\mathsf{tid}}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$.
     (f) $c_{\mathsf{tid},v} \leftarrow \mathsf{SKE}_1.\mathsf{Enc}(\mathsf{SKE}_1.\mathsf{sk}, (\widehat{F}_{\mathsf{tid},v}, \mathsf{FPFE.sk}_{H_{\mathsf{tid},v}}))$.
  2. Let $G_{\mathsf{tid},v} = G[\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, c_{\mathsf{tid},v}, \{c_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}}]$ as defined in Figure 5.
  3. $\mathsf{FE.sk}_{G_{\mathsf{tid},v}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_{\mathsf{tid},v})$.
  4. Send $\mathsf{SK}_{\mathsf{tid},v} = \mathsf{FE.sk}_{G_{\mathsf{tid},v}}$ to the adversary.
- **Message Query:** On message query (x_e, e):
  1. $\mathsf{FPFE.ct}_e \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (0^{\ell_{\mathcal{X}}}, e, 1, \mathsf{SKE}_2.\mathsf{sk}))$.
  2. $\mathsf{CT}_e = (\mathsf{FE.ct}, \mathsf{FPFE.ct}_e)$.
  3. Send CT_e to the adversary.
5. **Experiment Outcome**: $\mathcal{A}$ outputs a bit b' which is the output of the experiment.

Lemma 6.11. *If* PRF_3 *is a secure* PRF*, then for all PPT adversaries* $\mathcal{A}$*,*

$$\left|\Pr[\mathbf{Hybrid}_6^{\mathcal{A}}(1^\lambda)] - \Pr[\mathbf{Hybrid}_7^{\mathcal{A}}(1^\lambda)]\right| \leq \mathsf{negl}(\lambda).$$

Proof. Note that each $\mathsf{PRF}_3.K_{\mathsf{tid}}$ is no longer in the FPFE function keys. Thus, this lemma follows by a straightforward reduction to the security of the PRF schemes. □

Hybrid$_8^{\mathcal{A}}(1^\lambda)$: We keep track of the output values of the adversary's message and function queries, and simulate the MHGC using these output values.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs parameters $(1^\lambda, 1^{d_{\max}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.
2. **Setup:**
   (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
   (b) Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.
3. **Encryption Setup:**
   (a) $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$.
   (b) $\mathsf{SKE}_1.\mathsf{sk} \leftarrow \mathsf{SKE}_1.\mathsf{Setup}(1^\lambda)$.
   (c) $\mathsf{SKE}_2.\mathsf{sk} \leftarrow \mathsf{SKE}_2.\mathsf{Setup}(1^\lambda)$.
   (d) $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (0^{\ell_{\mathsf{FPFE.msk}}}, 0^{\ell_{\mathsf{PRF}_1} \cdot K}, 0^{\ell_{\mathsf{PRF}_2} \cdot K}, 1, \mathsf{SKE}_1.\mathsf{sk}))$.
4. The adversary can make a polynomial number of function and message queries in topological order.

- **Function Query:** On function query $(\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$:
  1. **Compute Output Values:**
     (a) For $e \in S_{\mathsf{in}}$ where $x_{\mathsf{tid},e}$ is not already defined:
        i. If $\mathsf{type}(e) = \mathsf{input}$ and x_e is defined (as a message query), set $x_{\mathsf{tid},e} = x_e$.
        ii. Else, set $x_{\mathsf{tid},e} = \perp$.
     (b) $(\{x_{\mathsf{tid},\alpha}\}_{\alpha \in S^{\mathsf{out}}_{\mathsf{out}}}, \{x_{\mathsf{tid},\beta}\}_{\beta \in S^{\mathsf{int}}_{\mathsf{out}}}) = f(\{x_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}})$.
  2. **Compute Hardwired Values:**
     (a) If this is the first query to tid, compute the following:
        i. Initialize a MHGC simulator $\mathsf{MHGC.Sim}_{\mathsf{tid}} = (\mathsf{MHGC.Sim}_{\mathsf{tid}}.\mathsf{Setup}, \mathsf{MHGC.Sim}_{\mathsf{tid}}.\mathsf{GarbInp}, \mathsf{MHGC.Sim}_{\mathsf{tid}}.\mathsf{GarbCirc})$.
        ii. Run $\mathsf{MHGC.Sim}_{\mathsf{tid}}.\mathsf{Setup}(1^\lambda, 1^{d_{\max}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.
     (b) For $e \in S_{\mathsf{in}}$,
        i. If x_e is already defined (as a message query),
           A. $\widehat{X}_{\mathsf{tid},e} \leftarrow \mathsf{MHGC.Sim}_{\mathsf{tid}}.\mathsf{GarbInp}(\mathsf{MHGC.msk}_{\mathsf{tid}}, e)$.
           B. $c_{\mathsf{tid},e} \leftarrow \mathsf{SKE}_2.\mathsf{Enc}(\mathsf{SKE}_2.\mathsf{sk}, \widehat{X}_{\mathsf{tid},e})$.
        ii. Else, $c_{\mathsf{tid},e} \leftarrow \{0,1\}^{\ell_{\mathsf{SKE}_2.\mathsf{ct}}}$.
     (c) Let $H_{\mathsf{tid},v} = H^*[\mathsf{tid}, v, S_{\mathsf{in}}, \{c_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}}]$ as defined in Figure 7.
     (d) $\mathsf{FPFE.sk}_{H_{\mathsf{tid},v}} \leftarrow \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H_{\mathsf{tid},v})$.
     (e) $\widehat{F}_{\mathsf{tid},v} \leftarrow \mathsf{MHGC.Sim}_{\mathsf{tid}}.\mathsf{GarbCirc}(\mathsf{MHGC.msk}_{\mathsf{tid}}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, \{x_{\mathsf{tid},\alpha}\}_{\alpha \in S^{\mathsf{out}}_{\mathsf{out}}})$.
     (f) $c_{\mathsf{tid},v} \leftarrow \mathsf{SKE}_1.\mathsf{Enc}(\mathsf{SKE}_1.\mathsf{sk}, (\widehat{F}_{\mathsf{tid},v}, \mathsf{FPFE.sk}_{H_{\mathsf{tid},v}}))$.
  3. Let $G_{\mathsf{tid},v} = G[\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, c_{\mathsf{tid},v}, \{c_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}}]$ as defined in Figure 5.
  4. $\mathsf{FE.sk}_{G_{\mathsf{tid},v}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_{\mathsf{tid},v})$.
  5. Send $\mathsf{SK}_{\mathsf{tid},v} = \mathsf{FE.sk}_{G_{\mathsf{tid},v}}$ to the adversary.

- **Message Query:** On message query (x_e, e):
  1. $\mathsf{FPFE.ct}_e \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (0^{\ell_x}, e, 1, \mathsf{SKE}_2.\mathsf{sk}))$.
  2. $\mathsf{CT}_e = (\mathsf{FE.ct}, \mathsf{FPFE.ct}_e)$.
  3. Send CT_e to the adversary.

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

Lemma 6.12. *If* MHGC *is a secure multi-hop garbled circuit scheme, then for all PPT adversaries* $\mathcal{A}$*,*

$$\left|\Pr[\mathbf{Hybrid}_7^{\mathcal{A}}(1^\lambda)] - \Pr[\mathbf{Hybrid}_8^{\mathcal{A}}(1^\lambda)]\right| \leq \mathsf{negl}(\lambda).$$

Proof. First, we remark that since the queries are asked in topological order, we will be able to compute all the output values of a function before the function is queried. The security follows by a reduction to the security of MHGC. □

Hybrid$_9^{\mathcal{A}}(1^\lambda)$: This is the ideal world written using an explicit simulator Sim. This hybrid is identical to the previous hybrid except that we now get the output values $\{x_{\mathsf{tid},\alpha}\}_{\alpha \in S^{\mathsf{out}}_{\mathsf{out}}}$ from the security game.

1. **Parameters**: The adversary $\mathcal{A}$ receives security parameter 1^λ, and outputs parameters $(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$. The simulator Sim receives $(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.
2. **Setup:** The simulator Sim does the following:
   (a) $(\mathsf{FE.mpk}, \mathsf{FE.msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
   (b) Send $\mathsf{MPK} = \mathsf{FE.mpk}$ to the adversary.
3. **Encryption Setup:** The simulator Sim does the following:
   (a) $\mathsf{FPFE.msk} \leftarrow \mathsf{FPFE.Setup}(1^\lambda)$.
   (b) $\mathsf{SKE}_1.\mathsf{sk} \leftarrow \mathsf{SKE}_1.\mathsf{Setup}(1^\lambda)$.
   (c) $\mathsf{SKE}_2.\mathsf{sk} \leftarrow \mathsf{SKE}_2.\mathsf{Setup}(1^\lambda)$.
   (d) $\mathsf{FE.ct} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}, (0^{\ell_{\mathsf{FPFE.msk}}}, 0^{\ell_{\mathsf{PRF}_1 \cdot K}}, 0^{\ell_{\mathsf{PRF}_2 \cdot K}}, 1, \mathsf{SKE}_1.\mathsf{sk}))$.
4. The adversary can make a polynomial number of function and message queries in topological order.

- **Function Query:** On function query $(\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}})$,
  We compute the following output values:
  1. For $e \in S_{\mathsf{in}}$ where $x_{\mathsf{tid},e}$ is not already defined:
     (a) If $\mathsf{type}(e) = \mathsf{input}$ and x_e is defined (as a message query), set $x_{\mathsf{tid},e} = x_e$.
     (b) Else, set $x_{\mathsf{tid},e} = \bot$.
  2. $(\{x_{\mathsf{tid},\alpha}\}_{\alpha \in S^{\mathsf{out}}_{\mathsf{out}}}, \{x_{\mathsf{tid},\beta}\}_{\beta \in S^{\mathsf{int}}_{\mathsf{out}}}) = f(\{x_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}})$.

  The simulator Sim then receives $(\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, \{x_\alpha\}_{\alpha \in S^{\mathsf{out}}_{\mathsf{out}}})$ and does the following:
  1. **Compute Hardwired Values:**
     (a) If this is the first query to tid, compute the following:
        i. Initialize a MHGC simulator
        $\mathsf{MHGC.Sim}_{\mathsf{tid}} = (\mathsf{MHGC.Sim}_{\mathsf{tid}}.\mathsf{Setup}, \mathsf{MHGC.Sim}_{\mathsf{tid}}.\mathsf{GarbInp}, \mathsf{MHGC.Sim}_{\mathsf{tid}}.\mathsf{GarbCirc})$.
        ii. Run $\mathsf{MHGC.Sim}_{\mathsf{tid}}.\mathsf{Setup}(1^\lambda, 1^{d_{\mathsf{max}}}, 1^{\ell_{\mathcal{F}}}, 1^{\ell_{\mathcal{X}}}, \mathsf{type})$.
     (b) For $e \in S_{\mathsf{in}}$,
        i. If (e) has been queried as a message query,
           A. $\widehat{X}_{\mathsf{tid},e} \leftarrow \mathsf{MHGC.Sim}_{\mathsf{tid}}.\mathsf{GarbInp}(\mathsf{MHGC.msk}_{\mathsf{tid}}, e)$.
           B. $c_{\mathsf{tid},e} \leftarrow \mathsf{SKE}_2.\mathsf{Enc}(\mathsf{SKE}_2.\mathsf{sk}, \widehat{X}_{\mathsf{tid},e})$.
        ii. Else, $c_{\mathsf{tid},e} \leftarrow \{0,1\}^{\ell_{\mathsf{SKE}_2.\mathsf{ct}}}$.
     (c) Let $H_{\mathsf{tid},v} = H^*[\mathsf{tid}, v, S_{\mathsf{in}}, \{c_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}}]$ as defined in Figure 7.
     (d) $\mathsf{FPFE.sk}_{H_{\mathsf{tid},v}} \leftarrow \mathsf{FPFE.KeyGen}(\mathsf{FPFE.msk}, H_{\mathsf{tid},v})$.
     (e) $\widehat{F}_{\mathsf{tid},v} \leftarrow \mathsf{MHGC.Sim}_{\mathsf{tid}}.\mathsf{GarbCirc}(\mathsf{MHGC.msk}_{\mathsf{tid}}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, \{x_{\mathsf{tid},\alpha}\}_{\alpha \in S^{\mathsf{out}}_{\mathsf{out}}})$.
     (f) $c_{\mathsf{tid},v} \leftarrow \mathsf{SKE}_1.\mathsf{Enc}(\mathsf{SKE}_1.\mathsf{sk}, (\widehat{F}_{\mathsf{tid},v}, \mathsf{FPFE.sk}_{H_{\mathsf{tid},v}}))$.
  2. Let $G_{\mathsf{tid},v} = G[\mathsf{tid}, f, v, S_{\mathsf{in}}, S_{\mathsf{out}}, c_{\mathsf{tid},v}, \{c_{\mathsf{tid},e}\}_{e \in S_{\mathsf{in}}}]$ as defined in Figure 5.

3. $\mathsf{FE.sk}_{G_{\mathsf{tid},v}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}, G_{\mathsf{tid},v})$.
4. Send $\mathsf{SK}_{\mathsf{tid},v} = \mathsf{FE.sk}_{G_{\mathsf{tid},v}}$ to the adversary.

- **Message Query:** On message query (x_e, e), the simulator Sim receives just (e) and does the following:
  1. $\mathsf{FPFE.ct}_e \leftarrow \mathsf{FPFE.Enc}(\mathsf{FPFE.msk}, (0^{\ell_x}, e, 1, \mathsf{SKE}_2.\mathsf{sk}))$.
  2. $\mathsf{CT}_e = (\mathsf{FE.ct}, \mathsf{FPFE.ct}_e)$.
  3. Send CT_e to the adversary.

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

Lemma 6.13. *For all adversaries $\mathcal{A}$,*

$$\left|\Pr[\mathbf{Hybrid}_8^{\mathcal{A}}(1^\lambda)] - \Pr[\mathbf{Hybrid}_9^{\mathcal{A}}(1^\lambda)]\right| = 0$$

Proof. The hybrids are identical. □

Thus, our lemmas give us the following corollary:

Corollary 6.14. *If*

- MHGC *is a secure mulit-hop garbling scheme,*
- $\mathsf{PRF}_1, \mathsf{PRF}_2, \mathsf{PRF}_3$ *are secure pseudorandom function families,*
- $\mathsf{SKE}_1, \mathsf{SKE}_2$ *are secure secret-key encryption scheme,*
- FE *is an adaptive-IND-secure public-key FE scheme for* $\mathsf{P/Poly}$*,*
- *and* FPFE *is a function-private, adaptive-IND-secure FE scheme for* $\mathsf{P/Poly}$*,*

then MHFE *is a one-ciphertext, SIM-secure public-key MHFE scheme for* $\mathsf{P/Poly}$*.*

Proof. The corollary immediately follows from Lemmas 6.5-6.13. □

Corollary 6.14 then implies Theorem 6.1 since we can instantiate each of the required primitives from an adaptive-IND-secure public-key FE scheme for $\mathsf{P/Poly}$.

7 Acknowledgements

This research was supported in part from a Simons Investigator Award, DARPA expMath award, Laude Moonshot grant, 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 HR001126CE054.

Very minor portions of this submission (mostly figures) were prepared with the assistance of generative AI tools.

8 References

[ABF+13] Joël Alwen, Manuel Barbosa, Pooya Farshim, Rosario Gennaro, S. Dov Gordon, Stefano Tessaro, and David A. Wilson. On the relationship between functional encryption, obfuscation, and fully homomorphic encryption. In Martijn Stam, editor, *14th IMA International Conference on Cryptography and Coding*, volume 8308 of *LNCS*, pages 65–84. Springer, Berlin, Heidelberg, December 2013.

[ACF+20] Shweta Agrawal, Michael Clear, Ophir Frieder, Sanjam Garg, Adam O'Neill, and Justin Thaler. Ad hoc multi-input functional encryption. In Thomas Vidick, editor, *ITCS 2020*, volume 151, pages 40:1–40:41. LIPIcs, January 2020.

[ACFQ22] Prabhanjan Ananth, Kai-Min Chung, Xiong Fan, and Luowen Qian. Collusion-resistant functional encryption for RAMs. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 160–194. Springer, Cham, December 2022.

[AGVW13] Shweta Agrawal, Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption: New perspectives and lower bounds. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 500–518. Springer, Berlin, Heidelberg, August 2013.

[AM18] Shweta Agrawal and Monosij Maitra. FE and iO for Turing machines from minimal assumptions. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 473–512. Springer, Cham, November 2018.

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

[AS17] Shweta Agrawal and Ishaan Preet Singh. Reusable garbled deterministic finite automata from learning with errors. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *ICALP 2017*, volume 80 of *LIPIcs*, pages 36:1–36:13. Schloss Dagstuhl, July 2017.

[AW17] Shashank Agrawal and David J. Wu. Functional encryption: Deterministic to randomized functions from simple assumptions. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 30–61. Springer, Cham, April / May 2017.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Shafi Goldwasser, editor, *ITCS 2012*, pages 326–349. ACM, January 2012.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Berlin, Heidelberg, August 2001.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 784–796. ACM Press, October 2012.

[BKMT21] Pramod Bhatotia, Markulf Kohlweiss, Lorenzo Martinico, and Yiannis Tselekounis. Steel: Composable hardware-based stateful and randomised functional encryption. In Juan Garay, editor, *PKC 2021, Part II*, volume 12711 of *LNCS*, pages 709–736. Springer, Cham, May 2021.

[BKS25] Kaartik Bhushan, Alexis Korb, and Amit Sahai. Dynamic bounded-collusion streaming functional encryption from minimal assumptions. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part III*, volume 16002 of *LNCS*, pages 137–169. Springer, Cham, August 2025.

[BRS13a] Dan Boneh, Ananth Raghunathan, and Gil Segev. Function-private identity-based encryption: Hiding the function in functional encryption. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 461–478. Springer, Berlin, Heidelberg, August 2013.

[BRS13b] Dan Boneh, Ananth Raghunathan, and Gil Segev. Function-private subspace-membership encryption and its applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 255–275. Springer, Berlin, Heidelberg, December 2013.

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

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

[CDG+18] Jérémy Chotard, Edouard Dufour Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Decentralized multi-client functional encryption for inner product. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 703–732. Springer, Cham, December 2018.

[CDSG+20] Jérémy Chotard, Edouard Dufour-Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Dynamic decentralized functional encryption. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 747–775. Springer, Cham, August 2020.

[DG17] Nico Döttling and Sanjam Garg. Identity-based encryption from the Diffie-Hellman assumption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 537–569. Springer, Cham, August 2017.

[DGKS25a] Pratish Datta, Jiaxin Guan, Alexis Korb, and Amit Sahai. Adaptively secure streaming functional encryption. In Benny Applebaum and Huijia (Rachel) Lin, editors, *TCC 2025, Part II*, volume 16269 of *LNCS*, pages 326–361. Springer, Cham, December 2025.

[DGKS25b] Pratish Datta, Jiaxin Guan, Alexis Korb, and Amit Sahai. (Multi-input) FE for randomized functionalities, revisited. In Benny Applebaum and Huijia (Rachel) Lin, editors, *TCC 2025, Part II*, volume 16269 of *LNCS*, pages 291–325. Springer, Cham, December 2025.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

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

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

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

[GHV10] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. i-Hop homomorphic encryption and rerandomizable Yao circuits. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 155–172. Springer, Berlin, Heidelberg, August 2010.

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

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

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

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

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

[HIJ+17] Shai Halevi, Yuval Ishai, Abhishek Jain, Ilan Komargodski, Amit Sahai, and Eylon Yogev. Non-interactive multiparty computation without correlated randomness. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 181–211. Springer, Cham, December 2017.

[HKM+24] Taiga Hiroka, Fuyuki Kitagawa, Tomoyuki Morimae, Ryo Nishimaki, Tapas Pal, and Takashi Yamakawa. Certified everlasting secure collusion-resistant functional encryption, and more. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part III*, volume 14653 of *LNCS*, pages 434–456. Springer, Cham, May 2024.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd ACM STOC*, pages 60–73. ACM Press, June 2021.

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

[KSY15] Ilan Komargodski, Gil Segev, and Eylon Yogev. Functional encryption for randomized functionalities in the private-key setting from minimal assumptions. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 352–377. Springer, Berlin, Heidelberg, March 2015.

[NR16] Jesper Buus Nielsen and Samuel Ranellucci. Reactive garbling: Foundation, instantiation, application. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 1022–1052. Springer, Berlin, Heidelberg, December 2016.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010.

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

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