SCALES

MPC with Small Clients and Larger Ephemeral Servers

Anasuya Acharya [1], Carmit Hazay [2], Vladimir Kolesnikov [3], and Manoj Prabhakaran[4]

[1] Bar Ilan University
acharya@biu.ac.il
[2] Bar Ilan University
carmit.hazay@biu.ac.il
[3] Georgia Institute of Technology
kolesnikov@gatech.edu
[4] Indian Institute of Technology Bombay
mp@cse.iitb.ac.in

Abstract. The recently proposed YOSO model is a groundbreaking approach to MPC, executable on a public blockchain, circumventing adaptive player corruption by hiding the corruption targets until they are worthless. Players are selected unpredictably from a large pool to perform MPC subtasks, in which each selected player sends a single message (and reveals their identity). While YOSO MPC has attractive asymptotic complexity, unfortunately, it is concretely prohibitively expensive due to the cost of its building blocks.

We propose a modification to the YOSO model that preserves resilience to adaptive server corruption, but allows for much more efficient protocols. In SCALES (Small Clients And Larger Ephemeral Servers) only the servers facilitating the MPC computation are ephemeral (unpredictably selected and "speak once"). Input providers (clients) publish problem instance and collect the output, but do not otherwise participate in computation. SCALES offers attractive features, and improves over YOSO in outsourcing MPC to a large pool of servers under adaptive corruption. We build SCALES from Rerandomizable Garbling Schemes (RGS). RGS is a contribution of independent interest with additional applications.

1 Introduction

A recent line of research, motivated by platforms such as blockchains, studies multi-party computation (MPC) with specialized communication and computation patterns [BGG+20, GHK+21, CGG+21, GMPS21]. While the specifics differ, these models leverage a dynamic pool of workers, unavailable throughout the protocol. Most excitingly, [BGG+20, GHK+21] show it is possible to only depend on *ephemeral* workers, who carry out some local computation, publish a *single* message on a bulletin board, and then vanish from the system. This is pithily captured in the name YOSO (You Only Speak Once) [GHK+21]. An attractive model for leveraging short-term workers, crucially, YOSO eliminates or drastically reduces the window for *adaptive corruption* of these workers. In particular, this for the first time enables efficient massive-scale MPC with adaptive corruption, achieved simply by delegating the computation to a small unpredictably selected YOSO subcommittee.

Even as the YOSO results [BGG+20, GHK+21] are powerful, they do leave room for improvement: they rely on strong honest-majority assumptions and expensive target-anonymous channels. Similarly, non-YOSO work requires honest majority [CGG+21] or complex setups, such as Conditional Storage and Retrieval in [GMPS21].

We propose an alternate model, where *light-weight input parties* participate in the initial and final stages of the protocol and do retain some state in between; but the bulk of the computation is carried out by *ephemeral servers* that are capable of performing computationally demanding tasks. Here, by 'light-weight', we mean that the complexity of each input does not depend on the function's complexity or inputs of other parties, but only on the size of its own inputs, and the number of participating ephemeral servers. There is no setup other than a bulletin board, and the corruption model allows all-but-one server participating in the computation to be corrupt, allowing for even very small numbers of servers. Moreover, by requiring the input parties to send a second message, we let them *control when the computation finishes* — arguably a desirable feature, especially when the number of servers used can be dynamic. Crucially, our ephemeral servers send a single message each, maintaining YOSO-like resilience to adaptive corruptions.

Note that a bulletin board is much simpler than target-anonymous channels in many ways. In particular, in a semi-honest setting, a bulletin board can be implemented by a *single* party, without requiring any honest majority assumptions, as there are no secrets to hide. But a target-anonymous channel would need more than a single honest party, and further if an efficient implementation involving a small committee is resorted to and the adversary can corrupt parties adaptively, a large honest majority is needed: $> 50\%$ [GHM+21] or $> 71\%$ [BGG+20].

We seek a protocol without complex setup and based only on standard cryptographic assumptions. Our solution builds on *rerandomizable* Garbled Circuits, formalized as Rerandomizable Garbling Schemes (RGS). In this work we shall focus on security against passive corruption.

1.1 Summary of Our Contributions

Before going further, we summarize the contributions in this work:

- *MPC with Small Clients and Larger Ephemeral Servers (SCALES).* Our main high-level contribution is the introduction of an attractive setting for MPC with ephemeral servers and limited interaction in Section 3. SCALES preserves YOSO-like resilience to adaptive server corruptions, and hence also allows outsourcing secure computation to blockchain (Section 1.2). We construct an efficient semi-honest SCALES protocol, where each server does work proportional to the circuit size, and each client proportional to its input size (Section 6).

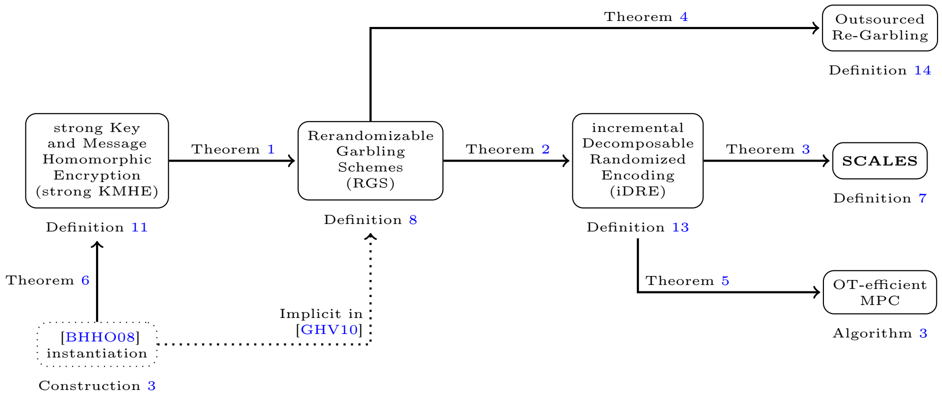

Fig. 1 Our contributions

- *Defining basic cryptographic primitives.* We formalize the following notions used in constructing a SCALES protocol, which we believe to be of independent interest, and investigate their relationship: 1) *Rerandomizable Garbling Scheme* (RGS) (Section 4), a generalization of Garbling Schemes (GS) to the setting of multiple garblers, each is *sequentially* involved in garbling, 2) *Strong Key-and-Message Homomorphic Encryption* (strong KMHE), and 3) A new multi-party notion of a randomized encoding, *incremental Decomposable Randomized Encoding* (iDRE) (Section 5).
- *Corresponding constructions.* We show that a construction of Boneh et al. [BHHO08], following the analysis in [NS09, GHV10], yields strong KMHE for a useful class of key and message transformations. Next, we show that such a strong KMHE scheme, when used as the encryption scheme in a version of garbled circuit (GC) yields an RGS. We then combine this RGS with a (weak) KMHE scheme, to obtain an iDRE scheme, which can be directly used for SCALES.
- *Further Applications.* Beyond being building blocks for protocols in the SCALES setting, RGS and iDRE are highly useful for other MPC settings as well.
  - **Outsourced Regarbling.** We show that an RGS directly yields an "Outsourced Regarbling" scheme. In a secure 2-party computation (2PC) setting, when Alice's (secret) function is to be securely evaluated on many inputs held by Bob, an outsourced re-garbling scheme allows Alice to outsource much of her work to a semi-honest server.
  - **Efficient MPC with optimal OT complexity.** An iDRE can be used to implement general n-party MPC protocols secure against a semi-honest corruption of $(n-1)$ parties. For an input size m, such a protocol takes $O(n \times m)$ string-OT calls, meeting the lower bound on OT complexity for this setting, as proven in [HIK07]. While [HIK07] also presents a protocol that meets this bound, their protocol requires OT strings to be of the size of the truth-table of the function being computed. In contrast, an iDRE-

based protocol (Section 7.2) runs OT of constant-size strings. Unlike [HIK07] which is in the information-theoretic OT-hybrid model, we do allow a single *black-box* invocation of iDRE. However, we note that invoking iDRE with each party carrying out at most one (re-)encoding step does not trivialize OTs: thanks to the sequential communication pattern, such an invocation of iDRE by itself would not provide a means to implement MPC without OTs or further computational assumptions.

- *Closing an analysis gap in previous work.* Rerandomizing GCs has previously been explored in the context of multi-hop homomorphic encryption by Gentry et al. [GHV10]. They define *rerandomizable SFE* (Secure Function Evaluation) and instantiate it using the encryption scheme of [BHHO08], though the specific security guarantees of strong KMHE were not identified there. Although their construction does satisfy their definition of rerandomizable SFE, their proof has a gap, which we point out. We also clarify that although [GHV10] uses similar building blocks, its multi-hop homomorphic encryption setting is inherently different from SCALES.

1.2 Our Main Contribution: SCALES MPC

The motivation for SCALES follows that of the recently proposed YOSO MPC. The YOSO (You Only Speak Once) property and model of MPC, introduced by Gentry et al. [GHK+21], requires that protocol participants each send a single message during the execution. Combined with known techniques for players to self-select at random for a task (cf. Bitcoin miners who self-select for proposing a block by finding a hash preimage of a special form), YOSO finally *offers hope* for efficient large-scale MPC in the setting with adaptive player corruption. Indeed, standard adaptively secure n-party MPC protocols have costs quadratic in n. In large-scale MPC, electing a small committee who will then evaluate the function on behalf of all n players is far more efficient, asymptotically and practically. Unfortunately, with adaptive corruptions, this breaks down, as adaptive adversary will simply corrupt all members of the committee (its corruption budget is a fraction of n, which is greater than the committee size). This is where YOSO saves the day: committee members are unidentifiable since they are self-selected and are removed from the committee as soon as they post a message, or "speak". Thus, an adaptive adversary does not know whom to corrupt until it is too late, and the committee executing the YOSO MPC is secure against adaptive corruptions. A particular application of interest of YOSO MPC is MPC over a blockchain, where blockchain nodes form the pool of MPC players, and inputs may come from participants such as accounts or wallets. Quite surprisingly, YOSO is achievable [GHK+21], despite numerous technical obstacles, such as the need for players executing i-th MPC round to send encrypted messages (e.g. containing internal state) to unidentified future round-$(i+1)$ committee members. Unfortunately, however, this protocol's costs are prohibitive for practice.

SCALES MPC motivation. Motivated by *practically efficient* YOSO-style large-scale MPC, and with a particular eye on outsourced MPC and blockchain

MPC, we introduce our SCALES (Small Clients And Larger Ephemeral Servers) MPC model. We keep the crucial YOSO property that servers speak once (and hence committee is protected against full dynamic corruption). Our clients (input providers) speak twice, to publish a problem instance and to collect the answer. This weakening of the model allows us to have a much more efficient instantiation than YOSO. We compare the two models in more detail in Section 1.5.

Syntactically, this is more permissive than YOSO; this is consistent with the goals of blockchain and outsourced MPC, and YOSO. Indeed, dynamic corruption of individual clients only threatens their security, and not of the computation and other clients. Essentially, YOSO's main advantage over SCALES is the ability to hide client identities, a less appealing feature that can still be added to SCALES by clients sending their state to future decoding players using expensive YOSO technique *once*. In return, we get a much higher performance as discussed in Sections 1.3 and 1.5 and several additional features. Note, we do not reduce computation *per server*, but rather total servers' work.

SCALES model. A set of *lightweight* input providers wish to securely compute a function of all their inputs. The bulk of the computation itself is *outsourced* to a pool of servers. We assume broadcast through a public bulletin board and that every message to be sent is posted onto it. In the computation, the set of input providers first post encoding of their inputs. Next, one by one, a server from the pool, upon turning online, reads the state of the bulletin board, performs specified computation, erases its state, posts its outcome, and goes offline. Once sufficiently many servers have been involved in the computation, the input providers post a second message based on the state of the bulletin board, and the decoding procedure can take place publicly using all the information posted[5].

SCALES features.

1. As in YOSO, the servers are speak-once and dynamically *self-selected*. Their identities are unknown until they have completed their part of the computation and erased their internal state. Hence they are not vulnerable to dynamic corruption.
2. The number of participating servers need not be fixed ahead of time. For instance, it can be based on a function of the (unpredictable) server identities.
3. The input parties need not interact with, or even be aware of, each other. Their complexity is independent of the number of other input players.
4. A SCALES protocol is also useful in settings with very few – say, two – non-colluding servers. We remark that while similar non-interactive outsourcing using GC has been considered [MRZ15], without rerandomization they require that the GC evaluator does not collude with *either* of the two servers.
5. An input provider could ensure that it is happy with the set of servers who have taken part in the protocol, before allowing the final decoding to proceed (by holding off from posting its second message).

[5] The final output of the protocol can easily be made private - known only to the clients. This is done by computing a function that gives an encryption of the desired output under the client's key.

6. In the case that more than one server posts a message in the same round, creating a fork in the computation, the input providers can choose which chain of server computations they want to recognize (by posting a second message only for that set of servers).

Further, one could add a requirement that the first message from the input parties be "reusable," in the spirit of recent two-round MPC protocols [BJKL21,BGSZ21]. We omit this from our definition for simplicity. However, this is satisfied by our construction that is based on a 2-round OT protocol with a reusable first message.

1.3 Other Contributions in More Detail

Rerandomizable Garbling Schemes. We formalize RGS as a powerful generalization of Garbling Schemes (GS) to the setting of multiple garblers. This deviates from the multi-party garbling of [BMR90] where all garblers symmetrically contribute to the final garbling. An RGS retains the standard garbling procedure Gb, and supplements it with an additional function Rerand. Given a garbling (without its input encoding function), Rerand rerandomizes it, producing a new garbling that is indistinguishable from a fresh garbling. Rerand also supplies a transformation that, when applied to the encoding function of the original garbling, will yield the encoding function of the regarbling.

The RGS approach allows the garblers to be ephemeral. Further, the number of garblers can be dynamically selected, if desired. The computation and communication complexity of garblers remain *constant* with the number of garblers, vs *quadratic* in the traditional approach.

Constructing a Rerandomizable Garbling Scheme. We provide an RGS construction based on GC [Yao86] that we endow with a secure regarbling procedure. To rerandomize GC, we follow [GHV10], where each output label is additively secret-shared into two shares, and each share is encrypted (with strong KMHE) under a single input label as key. This garbling variant is rerandomization-friendlier than the double-key encryption schemes used in standard versions of garbled circuits (e.g., [LP09]).

Our strong KMHE abstraction supports both key and message homomorphism, a property that is crucial for achieving private garbling rerandomization. In essence, rerandomization follows by transforming every garbled row into a fresh ciphertext, encrypting a new label share. To maintain consistency across garbled gates, we apply a corresponding transformation to wire labels.

RGS security requires that a fresh garbling is indistinguishable from a rerandomized one, even given randomness used in the initial GC. Somewhat informally, this property boils down to indistinguishability between a ciphertext that is either encrypted under a transformed key or a fresh independent key, even given the original key. This is the property needed to close the gap in the [GHV10] proof. We further prove that the scheme of [BHHO08] meets our security definition.

A SCALES Scheme. In a SCALES scheme, all servers must garble jointly to prevent a successful server-evaluator collusion. Our model requires that this is done in a sequential manner. We build SCALES protocol from RGS by letting the ephemeral servers play the role of the (re-)garblers, and output is obtained by evaluating the resulting GC. We must also securely apply the input encoding transformations generated by RGS. Regarblers can do this because we use KMHE as our encryption scheme. Finally, active input keys are obtained by clients by running OT with each of the garblers. This can be done to fit with our communication pattern. Our resulting protocol is secure against all-but-one corruption of the ephemeral garblers and, given an OT that is secure against adaptive corruption of receivers, our protocol also withstands adaptive corruption of a subset of the clients.

Performance. As SCALES approximates YOSO both in motivation and formalization, we focus on the YOSO comparison (simplified to the semi-honest setting, without considering their use of NIZKs). In SCALES, per client's input bit, his work to generate the first message (of total two) is constant; to generate the second message, client's work is proportional to the number of ephemeral servers. Unlike all previous YOSO work, the number of ephemeral servers required for SCALES, is arbitrary (as long as at least one of them is honest), and is independent of the computed functionality, allowing small client, as well as small total server cost. Further, unlike YOSO protocols, we do not require the use of expensive target-anonymous channels or even a PKI.

Our message and round complexity is significantly lower than in prior YOSO work. This is *crucial* for performance in the blockchain setting, as blockchain latency dominates the overall turn around time. We have a small number of messages posted, grouped into a smaller number of rounds (the clients post in parallel, and the number of servers can be as low as 2, depending on the trust assumptions – each server posting one message), while other works (YOSO and non-YOSO such as fluid MPC, [RS21], and others) are based on GMW/Beaver triples and have a number of rounds linear in the circuit depth, each one with a committee (whose size depends on the trust assumptions).

1.4 Future Work

We mention a few important directions as follow-up for our work. Firstly, our RGS construction's efficiency overheads (beyond a conventional garbled circuit) stem from the underlying strong KMHE scheme. While the scheme of Boneh et al. [BHHO08] happens to meet this new definition, it was not designed just for that. This leads to the question of designing a more efficient strong KMHE scheme, so as to reduce the overhead incurred by our RGS construction.

Secondly, in our SCALES construction, for the sake of simplicity, we restricted ourselves to the semi-honest setting. In a setting with a common reference string (CRS), full security can be readily achieved using generic NIZK proofs. However, given the specific nature of our protocol using RGS, it is plausible that cheaper cut-and-choose techniques or SNARGs can be used instead of generic NIZK.

Other alternatives, which may allow additional interaction could also be explored. We leave this for future work.

Furthermore, note that in SCALES, we require that the input providers run in sublinear time in $|C|$, the size of the circuit, and we may have only a constant number of servers. We leave it open to construct protocols where the servers are also sublinear in $|C|$. In such a case, we conjecture that the computation must be done in the public decoding phase. Even assuming that the servers are all fully trusted, this entails a form of randomized encoding, where not just the depth, but also the size of the encoding circuit is sublinear in the circuit size. This simplified problem roughly corresponds to 'succinct randomized encodings' [BCG+18], a primitive that entails indistinguishability obfuscation. The full problem (SCALES with corruptible servers, and all clients and servers being sub-linear in $|C|$) seems hard to solve even using iO.

We also leave open the question of whether there is an *information-theoretic* iDRE with sub-exponential communication complexity for an interesting class of functions? This has an important implication for a theoretical question studied by Harnik et al. [HIK07]. They showed that for n parties to compute a function against unlimited corruption with information-theoretic security using oblivious transfer (OT) channels, all pairs of parties should use at least one instance of OT between them. They matched this lower bound with a truth-table based construction, which requires exponential communication. We remark that an iDRE yields a solution to the same problem with essentially an optimal number of OTs, and hence an information-theoretic iDRE (for some family of functions) with polynomial, or even sub-exponential, communication complexity would improve [HIK07].

Finally, we leave it open to obtain alternate RGS constructions based on garbling schemes other than garbled circuits.

1.5 Related Work

Alternate MPC Models. Several recent works, many inspired by a blockchain-like setting, have considered MPC with specialized communication patterns. These models are generally incomparable with each other, and with SCALES. However, they do share some of the motivations and features of SCALES, and we briefly discuss them below. Table 1 summarizes some of the features discussed below.

You Only Speak Once (YOSO). As discussed in Section 1.2, our work is motivated by the YOSO model of MPC [BGG+20, GHK+21], which aims to eliminate the threat of adaptive corruptions by ensuring that the adversary does not know who the committee members are among *many* possible players, and hence cannot take advantage of its adaptive corruption power.

We consider a complementary MPC model that admits potentially more efficient solutions. We eliminate the need for expensive target-anonymous channels by requiring that each server accesses a bulletin board and sends a *single* message to it. Further, we permit a corrupted majority over *all participating servers*, whereas YOSO requires minority of corruptions *in each committee*, with threshold close to $t = 1/4$. At the same time, we keep the main attraction of YOSO: ephemeral servers that may securely self-select, and thus facilitate, MPC service in the presence of an adaptive adversary.

Construction	Adversary Type	Corruption Threshold	Adaptive Corruption	Ephemeral-Servers	Setup
YOSO [BGG+20] [GHK+21]	malicious	minority	Yes	Yes	Target-Anonymous Channels
Fluid MPC [CGG+21]	unbounded malicious	minority in each committee	No	No	Broadcast, Private Channels
Le Mans [RS21]	malicious	all-but-one in each committee	No	No	Broadcast, Private Channels
MPC on the Blockchain [GMPS21]	malicious	as in the underlying protocol	No	No	CSaR
SCALES Definition 7	semi-honest	all-but one server	Yes	Yes	Bulletin Board

Table 1 Related MPC committee-based protocols and a summary of their features.

As a trade off for better efficiency and larger corruption threshold, SCALES relies on a less constrained communication model than YOSO's: our input players speak twice. However, corrupting input player only results in compromise of that player's input. We believe this does not significantly weaken the applicability of the model: in practice, MPC input providers may be known to the adversary anyway. We outline conceptual performance improvements over prior YOSO protocols in Section 1.3.

We remark that while in this work we have limited ourselves to semi-honest SCALES, full security can be readily achieved using generic NIZK proofs, matching YOSO in this aspect. However, given the specific nature of our protocol using RGS, it is plausible that cheaper cut-and-choose techniques can be used instead of generic NIZK. We leave this for future work.

Blockchain-Enabled Non-Interactive MPC. Goyal et al. [GMPS21] explores blockchain-assisted MPC. Here input providers enjoy least-possible participation: they deposit input and garblings of an MPC protocol's next-message function into so-called conditional storage and retrieval systems (CSaRs). CSaRs' correct and secure operation is delegated to the blockchain. Then the blockchain executes the MPC protocol at its leisure by processing the garbled next-message functions. In contrast, our motivating application is MPC computation on the blockchain performed by a committee of servers, which the adversary is unable to adaptively corrupt. While our communication model is more constrained, our solution is far more practical and only requires a bulletin board; [GMPS21] should be viewed as a fundamental feasibility result.

Fluid-MPC. Fluid MPC [CGG+21] allows parties to dynamically join and leave the computation. These parties are designated by a computing committee, whose membership itself evolves. It keeps and evolves the state of an MPC instance, eventually obtaining the output. Fluid MPC is a practical protocol, which relies on a strong corruption assumption: the adversary can corrupt only a minority of the servers in each committee. In contrast, in our motivating application, we aim to frustrate adaptive corruption of committee members by ensuring they only speak once.

A recent work [RS21] extends Fluid MPC to the dishonest majority setting. Crucially, [RS21] still does not meet the YOSO speak-once requirement. We note other costs of [RS21] (e.g., the number of epochs proportional to the size of the function) that we avoid.

Distributed Garbling Schemes. The RGS-based protocol for SCALES can be viewed as distributed garbling with crucial special properties needed for our application: (1) each garbler posts one message, and (2) unidirectional communication among garblers. We achieve this without preprocessing or correlated randomness. Previous distributed garbling protocols do not offer these properties, even given correlated randomness, e.g., authenticated triples.

Two-round MPC. It is also instructive to compare SCALES with 2-round MPC [GGHR14, GS18, BL18, BJKL21, BGSZ21]. The latter also involves input parties posting two rounds of messages to a bulletin board, based on which the output can be publicly computed. However, there the input parties incur communication and computation costs proportional to the entire circuit size of the function (in fact, the circuit size of an MPC protocol for the function). SCALES could be thought of as allowing ephemeral servers to process the bulletin board between the two rounds, so that the computational costs of the input parties becomes only proportional to the size of their own inputs.

Further, while not part of our formal definition, the SCALES setting can be extended to require the first message from the input players to be "reusable," a feature explored in the recent works on 2-round MPC [BJKL21, BGSZ21]. Our RGS-based construction already meets this additional requirement, at no additional cost.

Where efficiency of our protocols is concerned, note that we require security in the dishonest majority setting and so the concrete efficiency of our SCALES protocol is incomparable to that of previous work in the honest majority setting (YOSO, Fluid-MPC, etc.). Additionaly, note that although the servers sequentially perform computation only after the previous server has posted a message, the local actions of each server during rerandomizing are highly parallelizable: the server chooses a homomorphic function for each circuit wire independently, and each garbled gate can be rerandomized independently.

Randomized encodings. The abstraction of randomized encodings was introduced in [IK00], and has found a host of applications. A garbled circuit (GC) is a randomized encoding with desirable properties that were exploited in works such as [BMR90]. We mention the following constructions that are somewhat similar to iDRE introduced in this work.

- **Multi-party randomized encodings.** A notion of randomized encoding generated by multiple parties has been considered in the literature: [ABT18] proposed *Multi-Party Randomized Encoding* (MPRE). As in the case of iDRE, MPRE considers a distributed encoding of $f(x_1, \ldots, x_n)$. It uses many random strings, with the property that revealing a subset of these random strings will keep the other inputs hidden. A crucial distinction between iDRE and MPRE is that there is a protected part of the randomness in MPRE that must not be revealed at all. This is adequate for honest majority MPC, the main application in [ABT18], as this protected randomness remains secret-shared. In iDRE, there is no protected randomness, and all-but-one party could be corrupt. The two primitives also differ in several other ways, as their goals are quite different (reducing rounds in honest majority-MPC, in the case of MPRE, versus reducing the number of OTs in MPC with unrestricted collusion, in the case of iDRE).
- **Multi-hop homomorphic encryption.** Gentry et al. in [GHV10] introduced *multi-hop homomorphic encryption*. Setting aside the formulation as an encryption (which requires a rerandomizable 2-round OT protocol to be interpreted as an encryption process), their construction involved a set of servers jointly creating a garbled circuit. A crucial difference from the MPC setting is that an adversary who corrupts a subset of the players including the final evaluator would be able to learn much more about the individual inputs than just the final output. Nevertheless, a key tool used in this work – rerandomizable garbled circuits – turns out to be useful in our work. Though the specific manner in which garbled circuit rerandomization is defined and used by [GHV10] is not adequate for our purposes, we follow their approach of using a key-and-message-homomorphic encryption to implement it.

1.6 Technical Overview

We define and realize a new notion of randomized encodings [IK00] (Definition 5), the iDRE. This is the key construction underlying our SCALES protocol. For concreteness and simplicity, we first discuss our approach in the terminology of garbling schemes [BHR12], before casting it in terms of randomized encodings.

To be cast as a SCALES protocol, informally, our goal is minimally interactive multi-party circuit garbling. Therefore, we do not follow the constant-round BMR approach [BMR90], but instead explore *GC rerandomization*. This is a mechanism where an initial garbler generates a GC and each subsequent re-garbler re-randomizes the previous circuit and the labels. Breaking the connection between the labels of the garbled circuit and its regarbling, will allow for security in the presence of all-but-one corruption: indeed, even a single honest rerandomization will (if done right - we pay careful attention to precisely defining security requirements here) result in a GC where none of the generators knows the secrets completely (we get GC correctness "for free" in the semi-honest model).

Informally, a re-randomized garbled circuit $\hat{\mathcal{C}}'$ should allow the evaluation of a circuit C, where neither the garbler nor regarbler individually knows the correspondence between the labels and the actual wire values; the wire labels of the resulting garbled circuit $\hat{\mathcal{C}}'$ are effectively secret shared between them. To evaluate $\hat{\mathcal{C}}'$, each party $\mathcal{P}$ with an input bit (aka, an input party) picks up the shares of its input wire labels from the garblers (e.g., via OT), reconstructs them, and uses them for the evaluation. To violate input privacy, the evaluator would need to collude with *all* the garblers.

Rerandomizable Garbled Circuits from strong KMHE. Our main technical challenge was to design a garbling scheme that supports garbling rerandomization. We demonstrate how this can be achieved based on a strong key-and-message-homomorphic encryption (strong KMHE) scheme. We formalize a strong KMHE scheme as an encryption scheme[6] that permits transforming the key and/or the message in a ciphertext to obtain fresh-looking ciphertexts. Even a party who knows the original ciphertext's key should not be able to distinguish the result of randomly transforming the key from a fresh ciphertext using a fresh key. This is required to hold, even when given some leakage on the key transformation, in the form of a different input-output pair of the transformation. For our purposes, the message and key spaces would be the same, and the space of transformations supported for the two will be the same as well; these transformations will be linear. The specific instantiation of a strong KMHE scheme we use was constructed by Boneh et al. for a different purpose [BHHO08], and was shown to be leakage resilient by Naor and Segev [NS09]; further this scheme was used in [GHV10] for constructing a somewhat related task, rerandomizable secure function evaluation (or SFE), but without abstracting out the security properties we need.

We briefly sketch our construction of rerandomizable garbling schemes given a strong KMHE scheme. We view a garbled circuit as a collection of garbled gates where each gate consists of four ciphertexts, each requires a pair of keys to decrypt. However, instead of implementing a double-encryption scheme, as in standard garbling schemes, we additively share the plaintexts and encrypt each share using a single key. Therefore, each garbled row contains a *pair* of ciphertexts, encrypted under a single input key. (see Section 4).

To rerandomize a gate, the re-garbler R homomorphically alters each ciphertext, such that the result is a (new share of the) new output label encrypted under a new input key label. At a high level, we achieve this as follows. For each wire w_i, R first chooses a transformation σ_i that maps the space of the wire labels to itself. R's goal is to re-randomize each gate to enable correct evaluation. We do this by applying a sequence of homomorphic operations to

[6] We define this notion as a symmetric key primitive which suffices for our purposes. Nevertheless, the instantiation we give uses a public key encryption scheme [BHHO08].

(each element of) each garbled row, encrypted using strong KMHE: (1) update the plaintext using a transformation σ_g for the output wire of gate g and (2) update the key using a transformation σ_i for the input wire w_i. Furthermore, the homomorphic operations we use are linear. This ensures that applying the above to the ciphertexts encrypting secret shares of the output label will allow for the reconstruction of the new rerandomized label: σ_g applied to the old output label. To prevent a colluding G and E learning extra information, we require that the rerandomized garbled circuit $\hat{\mathcal{C}}'$ together with active input wire labels reveals no additional information. As a final step for rerandomization, the new 4-tuple of garbled rows is permuted.

Depending on how strong KMHE is instantiated, there are different tweaks that let the evaluator know which row of the garbled gate, when decrypted, gives a correct label. One such way would be to append a known prefix to the message labels that are encrypted. Care should be taken that during rerandomizing, the message domain operations do not affect this message prefix. The [BHHO08] instantiation for strong KMHE, explained next, supports such operations.

Strong KMHE instantiation. The encryption scheme of [BHHO08] can be used to instantiate strong KMHE in the computational setting under the Decisional Diffie-Hellman (DDH) hardness assumption. It allows homomorphic operations in both the key and plaintext domains and has the property that a transformed ciphertext is indistinguishable from a freshly encrypted ciphertext. For our purposes, and similarly in [GHV10], the key and plaintext domains are identical and amount to the set of balanced binary strings. Similarly, the key and plaintext domains are identical and correspond to the set of permutations. In order to differentiate a correct decryption during evaluation, this construction allows padding the plaintext label shares with an all-zero string. During rerandomizing, this prefix is always mapped onto itself. During evaluation, each garbled row is decrypted and the row yielding plaintexts padded with all-zero strings indicates the correct output label shares. We point the reader to Appendix A for more details.

Casting as a randomized encoding. For generality, we use this approach to describe a variant of a randomized encoding (Section 5). W.l.o.g., consider parties providing a single input bit each. We separate the role of parties $\mathcal{P} = (P_1, \cdots, P_m)$ providing input bits $x_1, \cdots, x_m$ from the role of *encoders* $\mathcal{E} = (E_1, \cdots, E_d)$ creating the randomized encoding. A garbled circuit presented above can be cast as a decomposable randomized encoding (DRE) $\hat{f}(x, r) = (\hat{f}_0(r), \hat{f}_1(x_1, r), \cdots, \hat{f}_m(x_m, r))$, where part of the encoding $\hat{f}_0(r)$ is independent of the input (and corresponds to the garbled circuit itself), and each $\hat{f}_i(x_i, r)$ depends on a bit x_i of the input (corresponding to the input labels).

Let $r = (r_1, \cdots, r_d)$ be the total randomness where encoder E_j possesses r_j. Each E_j creates values that act as shares of $\hat{f}_i(x_i, r)$ for both possible values of each $x_i \in \{0, 1\}$. Then each input party $P_i \in \mathcal{P}$ upon concluding an OT with each encoder, receives all these shares of $\hat{f}_i(x_i, r)$. E_1 uses r_1 to initiate the creation of $\hat{f}_0(r)$ similarly to G above. E_1 also incorporates encodings of the shares of each $\hat{f}_i(x_i, r)$ that it created, hence initiating the creation of a *final share* s_i. E_1 passes its initial $\hat{f}_0(r)$ and all such s_i to E_2. In turn, E_2 uses $r_2 \in r$ to rerandomize the initial $\hat{f}_0(r)$ it received, augments each s_i, and passes it on. This incremental process continues and the last encoder E_d hands the completed $\hat{f}_0(r)$ to the decoder D. Each value s_i is given to the corresponding input party $P_i \in \mathcal{P}$. These final shares are such that s_i, when combined with all the initial shares from the OT phase, gives $\hat{f}_i(x_i, r)$. This is reconstructed and sent to D. D decodes the complete DRE and receives the output. We denote our abstracted object by *incremental Decomposable Randomized Encoding* to highlight the incremental nature in which the DRE is created. A construction for this object directly implies a SCALES protocol.

2 Preliminaries

Circuit notation. For a function $f : \{0,1\}^m \to \{0,1\}^l$, a boolean circuit that computes it is denoted by $C = (\mathcal{W}, I, O, \mathcal{G})$. $\mathcal{W}$ is the set of all wires and $I \subset \mathcal{W}$ and $O \subset \mathcal{W}$ are the set of input and output wires respectively. Within $\mathcal{W}$, $I = (w_1, \cdots, w_m)$ are the m input wires, $w_{m+1}, \cdots, w_{m+p}$ are the p internal wires, and $O = (w_{m+p+1}, \cdots, w_{m+p+l})$ are the l output wires. These make $v = m+p+l$ total wires. $\mathcal{G} = (g_{m+1}, ..., g_{m+q})$ is the set of gates. Each $g_i = (w_\ell, w_r, w_i, op)$ is a binary gate where w_ℓ and w_r are the left and right input wires respectively, w_i is the output wire (uniquely defined by the gate index), and op represents the gate functionality (AND, XOR, etc.).

We consider the following notions of indistinguishability in our definitions:

Definition 1. *Two probability ensembles $X = \{X_n\}_{n\in\mathbb{N}}$ and $Y = \{Y_n\}_{n\in\mathbb{N}}$ defined over a finite domain D are* ***statistically indistinguishable****, denoted $X \overset{s}{\approx} Y$, if every positive polynomial $p(\cdot)$ and all sufficiently large n's,*

$$\Delta(X_n, Y_n) < \frac{1}{p(n)}$$

where,

$$\Delta(X_n, Y_n) = \frac{1}{2} \cdot \sum_{\alpha \in D} |\Pr[X_n = \alpha] - \Pr[Y_n = \alpha]|$$

Definition 2. *Two probability ensembles $X = \{X_n\}_{n\in\mathbb{N}}$ and $Y = \{Y_n\}_{n\in\mathbb{N}}$ are* ***computationally indistinguishable****, denoted $X \overset{c}{\approx} Y$, if for every PPT distinguisher D, every positive polynomial $p(\cdot)$ and all sufficiently large n's,*

$$|\Pr[D(X_n, 1^n) = 1] - \Pr[D(Y_n, 1^n) = 1]| < \frac{1}{p(n)}$$

2.1 Garbled Circuits

Garbling Schemes. We recall the notion of a garbling scheme abstracted in [BHR12] and simplify it for our use. That is, a garbling scheme is a tuple of algorithms

$\mathsf{GS} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev})$ where the probabilistic garbling algorithm Gb takes the function description f and outputs a garbled representation F and an input encoding function e. The deterministic input encoding algorithm En gets e and the function input x; and returns a garbled input representation X. Finally, the deterministic evaluation algorithm Ev takes F and X and outputs $f(x)$ by evaluating the garbling.

For simplicity, we limit the security properties of a garbling scheme to just *correctness* and *privacy* (and correspondingly, omit the separation between evaluation and "decoding" in [BHR12]). More formally,

Definition 3. *A* ***Garbling Scheme*** *for a function family $\mathcal{F}$ with input domain $\mathcal{X}$, and a leakage function $\phi : \mathcal{F} \to \{0,1\}^*$, is a tuple $\mathsf{GS} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev})$ of PPT algorithms, satisfying the following properties:*

– ***Correctness****: For every $f \in \mathcal{F}$ and input $x \in \mathcal{X}$,*

$$\Pr[y = f(x) \ : (F, e) \leftarrow \mathsf{Gb}(f),\ X = \mathsf{En}(e, x),\ y = \mathsf{Ev}(F, X)] = 1$$

– ***Privacy****: For all functions $f_0, f_1 \in \mathcal{F}$ such that $\phi(f_0) = \phi(f_1)$, and every $x_0, x_1 \in \mathcal{X}$ such that $f_0(x_0) = f_1(x_1)$,*

$$\{F_0, X_0\}_{(F_0,e_0)\leftarrow \mathsf{Gb}(f_0), X_0=\mathsf{En}(e_0,x_0)} \stackrel{c}{\approx} \{F_1, X_1\}_{(F_1,e_1)\leftarrow \mathsf{Gb}(f_1), X_1=\mathsf{En}(e_1,x_1)}$$

The above distribution ensembles are indexed by a security parameter κ that is an implicit input to Gb. When we need to make the randomness used by Gb explicit, we write it as an additional input, namely as $\mathsf{Gb}(f; r)$.

A special case of the above, a *projective garbling scheme* [BHR12] is a variant of garbling schemes whose input encoding function En is *projective*.

Definition 4. *A* ***Projective Garbling Scheme*** *for a function family $\mathcal{F}$ with input domain $\{0,1\}^m$, is a tuple $\mathsf{GS} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev})$ of PPT algorithms, such that GS is a garbling scheme (Definition 3) for $\mathcal{F}$ and the encoding function $\mathsf{En} : \{0,1\}^m \times \mathcal{E} \to \mathcal{Z}^m$ is such that $\forall x, x' \in \{0,1\}^m$ and $\forall e \in \mathcal{E}$, $\mathsf{En}(x, e) = (L_1, \cdots, L_m)$ and $\mathsf{En}(x', e) = (L_1^{'}, \cdots, L_m^{'})$ such that $\forall i \in [m]$, if $x_i = x_i'$ then $L_i = L_i^{'}$.*

Our construction employs *projective* garbling schemes. Looking ahead, we extend Definition 3 to a Rerandomizable Garbling Scheme (RGS) and instantiate it with rerandomizable GCs.

2.2 Randomized Encodings

A *Randomized Encoding*, defined in [IK00], is as follows:

Definition 5. *Let $X, Y, \hat{Y}, R$ be finite sets and let $f : X \to Y$. A function $\hat{f} : X \times R \to \hat{Y}$ is a* ***Randomized Encoding*** *of f, if it satisfies:*

– ***Correctness****: There exists a function Dec, a decoder, $\forall x \in X, r \in R$,*

$$\mathsf{Dec}(\hat{f}(x; r)) = f(x)$$

– ***Privacy**: There exists a randomized function* Sim*, a simulator,* $\forall x \in X$*,*

$$\{\mathsf{Sim}(f(x))\} \overset{c}{\approx} \{\hat{f}(x;r)\}_{r \in R}$$

We require that $\hat{f}$ is efficiently derivable from f using the function Enc, and that Dec and Sim are PPT. A variant of the above, a *Decomposable Randomized Encoding* (DRE), is defined as follows:

Definition 6. *For* $f : X_1 \times \cdots \times X_m \to Y$*, where* $\forall i \in [m], X_i = \{0,1\}$*, a* ***Decomposable Randomized Encoding*** *is a Randomized Encoding (Definition 5) of* f *with the form:*

$$\hat{f}((x_1, \cdots, x_m); r) = (\hat{f}_0(r), \hat{f}_1(x_1; r), \cdots, \hat{f}_m(x_m; r))$$

In a decomposable randomized encoding, each part of the encoding can depend on at most one input bit. It is well known that a projective garbling scheme (Definition 4) is a DRE. Looking ahead, we extend Definition 6 to an incremental Decomposable Randomized Encoding (iDRE) and instantiate it using a projective RGS.

2.3 Oblivious Transfer

Oblivious Transfer (OT) is a two party functionality between a sender S and a receiver R defined by $(e_b, \perp) \leftarrow \mathsf{OT}(b, (e_0, e_1))$. Our protocol in the SCALES model (Section 6), requires a 2-round OT protocol (with semi-honest, adaptive-receiver security). We denote this by the set of algorithms $\Pi^{\mathsf{OT}} = (\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_{out})$. The protocol starts by R computing $(m_1, \mathsf{Aux}) \leftarrow \mathsf{OT}_1(b)$ and sending the first OT message m_1 to S. Next, S computes the second OT message $m_2 \leftarrow \mathsf{OT}_2(m_1, (e_0, e_1))$ that is sent to R. Finally, R computes its output via $e_b \leftarrow \mathsf{OT}_{out}(\mathsf{Aux}, m_2)$.

We require that Π^{OT} be secure in the presence of a semi-honest adversary that statically corrupts S and adaptively corrupts R. This corruption can, in particular, take place after R sends m_1. In this case, the simulator is required to produce randomness t that is consistent with OT_1 upon corrupting R and learning its choice bit b.

We provide two definitions of the simulator, based on the corruption of S. First, for both cases, the OT first message is generated as $(m_1, \mathsf{state}) \leftarrow \mathsf{Sim}_1^{\mathsf{OT}}(\cdot)$. In the case that S is honest (and R is adaptively corrupted), the OT simulator is consists of the following two additional functions: $(m_2, \mathsf{state}') \leftarrow \mathsf{Sim}_2^{\mathsf{OT}}(m_1, \mathsf{state})$ and $t \leftarrow \mathsf{Sim}_3^{\mathsf{OT}}(\mathsf{state}', b, e_b)$. In the case that S is corrupted (and R is adaptively corrupted), the OT simulator is consists of only of the later algorithm $t \leftarrow \mathsf{Sim}_3^{\mathsf{OT}}(\mathsf{state}', b, e_b)$.

We instantiate Oblivious Transfer with a two-round protocol that is based on public key encryption schemes with an oblivious choice of the public key. Namely, an honest R picks pk_{1-b} obliviously while properly picking pk_b together with the matching secret key sk_b. It forwards these two public keys to S, receiving back two ciphertexts c_0 and c_1, respectively encrypting e_0 and e_1. R then uses sk_b to decrypt c_b.

In the adaptive simulation of OT_1, the simulator chooses both public keys with the knowledge of the secret key and later, upon corrupting R, declares that it chose the "right" key obliviously. Security follows here based on the obliviousness property of the underlying public key scheme. In case S is honest, the simulator needs to emulate the second OT message as well. In this case it cannot simply send two ciphertexts as it does not know the content of e_b (this is made public to it only upon corrupting R). We therefore use a non-committing encryption scheme [CFGN96] to generate these two ciphertexts.[7]

3 MPC with Small Clients and Larger Ephemeral Servers

We define a model, MPC with Small Clients and Larger Ephemeral Servers (SCALES), that is inspired by considerations that also underlie recent models like YOSO [BGG+20, GHK+21] and MPC on a blockchain [GMPS21]. Our goal is to achieve secure MPC in a setting where a set of light-weight input providers take the help of a dynamic set of stateless workers or *ephemeral servers*. The entire process involves communication only over a public bulletin board, and takes this form:

1. Initially, each input player posts a message on the bulletin board.
2. For as many iterations as desired, an ephemeral server is dynamically activated, which reads the bulletin board, carries out some local computation, erases its state, and posts a message back on the bulletin board. This computation may be proportional to size of the computed functionality.
3. Each input player reads the bulletin board (in parallel), and posts back another message on the bulletin board. These light weight parties' work is proportional to their input size times the number of ephemeral servers.
4. The output can be computed publicly based on the information in the bulletin board, implemented by another ephemeral server.

We shall require that the amount of computation and communication by each input player is proportional to its number of input bits, *independent of the size of the overall computation, or even the size of the overall input*. The communication constraints apart, we require the above to meet a standard security definition for MPC, against an adversary who can corrupt any subset of input players (possibly adaptively) and *all but one server*. As each server posts a single message before being erased, we shall consider only security against static corruption of servers (since a server's state is erased before it has started posting its message on the bulletin board). In this work, we focus on security against semi-honest corruption.

[7] An encryption scheme is non-committing if it can generate a dummy ciphertext that is indistinguishable from a real one. This can later be decrypted to any plaintext by producing an appropriate secret key decrypting the ciphertext to this plaintext. [YKT19, BBD+20] provide non-committing encryption schemes under the DDH assumption. The latter construction further achieves constant rate.

Definition 7. *A scheme for* ***MPC with Small Clients and Larger Ephemeral Servers*** *(SCALES) for a function family $\mathcal{F}$ over $\{0,1\}^m$ is a tuple of PPT algorithms* ($\mathsf{InpEnc}, \mathsf{FEnc}, \mathsf{Aggregate}, \mathsf{Decode}$) *such that the following random variables are defined as a function of $f \in \mathcal{F}$ and $x \in \{0,1\}^m$ (where R and T denote random-tape spaces for* FEnc *and* InpEnc *respectively):*

$$\begin{aligned}
r_j &\leftarrow R, t_i \leftarrow T && \forall j \in [d], i \in [m] \\
(z_i, w_i) &\leftarrow \mathsf{InpEnc}(x_i; t_i) && \forall i \in [m] \\
\mathcal{B}_j &\leftarrow \begin{cases} (f, \{z_i\}_{i \in m}) & \text{for } j = 1 \\ (\mathcal{B}_{j-1}, \mathsf{FEnc}(\mathcal{B}_{j-1}; r_j)) & \text{for } 1 < j \leq d \end{cases} \\
y_i &\leftarrow \mathsf{Aggregate}(\mathcal{B}_d, w_i) && \forall i \in [m].
\end{aligned}$$

Then the following properties hold:

– ***Correctness:*** *$\forall x = (x_1, \cdots, x_m) \in \{0,1\}^m$ and $d \in \mathbb{N}$,*

$$\Pr[\mathsf{Decode}(\mathcal{B}_d, \{y_i\}_{i \in [m]}) = f(x)] = 1$$

where $\forall j \in [d], \mathcal{B}_j = (\{z_i\}_{i \in [m]}, \{\alpha_k\}_{k \in [j]})$.

– ***Privacy:*** *There exists a 2-stage PPT simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ such that, $\forall f \in \mathcal{F}, x \in \{0,1\}^m$, $j^* \in [d]$, and $\mathcal{A}_1, \mathcal{A}_2 \subseteq [m]$,*

$$\begin{aligned}
(\alpha, \mathsf{Aux}) &\leftarrow \mathsf{Sim}_1(f, f(x), j^*, \{x_i\}_{i \in \mathcal{A}_1}) \\
\beta &\leftarrow \mathsf{Sim}_2(\mathsf{Aux}, \{x_i\}_{i \in \mathcal{A}_2}).
\end{aligned}$$

It holds that,

$$\begin{aligned}
\{\alpha\} &\overset{c}{\approx} \{\mathcal{B}_d, \{y_i\}_{i \in [m]}, \{r_j\}_{j \in [d] \setminus \{j^*\}}, \{t_i\}_{i \in \mathcal{A}_1}\} \\
\{\alpha, \beta\} &\overset{c}{\approx} \{\mathcal{B}_d, \{y_i\}_{i \in [m]}, \{r_j\}_{j \in [d] \setminus \{j^*\}}, \{t_i\}_{i \in \mathcal{A}_1}, \{t_i\}_{i \in \mathcal{A}_2}\}
\end{aligned}$$

Complexity. For simplicity, we have stated the definition without including any complexity requirements. To formalize the complexity requirement, we consider the functions in $\mathcal{F}$ as parameterized by a size parameter k, as $f_k : \{0,1\}^{m(k)} \to \{0,1\}^{q(k)}$, so that f_k has a circuit of size polynomial in k. Then, the algorithms InpEnc and $\mathsf{Aggregate}$ are required to be independent of k (but may depend on the security parameter κ)[8]. This requirement on the complexity of InpEnc and $\mathsf{Aggregate}$ is an important aspect of a SCALES protocol.

In a SCALES protocol, first, each input player runs the algorithm InpEnc and posts z_i on the bulletin board $\mathcal{B}$ (Step 1). Next, for each round j, each ephemeral server (as in Step 2) runs FEnc in the present state of the bulletin board $\mathcal{B}_{j-1}$

[8] Note that $\mathcal{B}_d$ has been specified as an input to $\mathsf{Aggregate}$, but $\mathsf{Aggregate}$ is required to only use a part of $\mathcal{B}_d$ which is independent of k.

and posts a message α_j on the board. After enough number of such iterations, each input player run $\mathsf{Aggregate}$ (Step 3) and post a message y_i. Finally, the function output is publicly derived using Decode (Step 4). The *privacy* guarantee requires that an adversary can corrupt all but the server indexed $j^* \in [d]$. It may corrupt an initial subset $\mathcal{A}_1 \in [m]$ of clients and between the first and the second time the clients speak, it can adaptively corrupt an additional set of $\mathcal{A}_2 \in [m]$ clients. Even in such a scenario, the view of the adversary needs to be simulatable. Building towards a protocol in the SCALES setting, we now define and construct our key building blocks.

4 Rerandomizable Garbling Schemes

In this section we define *Rerandomizable Garbling Schemes* (RGS) and construct such a scheme (Section 4.3) using a strong Key and Message Homomorphic Encryption scheme (strong KMHE - Section 4.1). Loosely speaking, a rerandomizable garbling scheme allows us to take a garbled representation F of a function and transform it into another garbled representation F' for the same function. This is done in such a way that it is impossible for a PPT distinguisher, given all the randomness used for garbling F, to distinguish F' from a fresh garbling of the function.

Formally, an RGS is a GS with an additional PPT algorithm $(F', \pi_{\mathsf{En}}) \leftarrow \mathsf{Rerand}(F)$ that outputs a rerandomized garbling F' and a transformation π_{En} to be applied on e such that the new encoding X', derived from applying En to $\pi_{\mathsf{En}}(e)$, when used with F', decodes correctly to $f(x)$. The security of RGS is captured by an additional property denoted by Rerand-*privacy* that is formalized as follows:

Definition 8. *A **Rerandomizable Garbling Scheme** for a function family $\mathcal{F}$ is a tuple of PPT algorithms $\mathsf{GS}' = (\mathsf{Gb}, \mathsf{Rerand}, \mathsf{En}, \mathsf{Ev})$ where, $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev})$ is a garbling scheme (Definition 3) for $\mathcal{F}$, and Rerand is a PPT algorithm such that the following is satisfied:*

– Rerand-***Privacy****: For every $f \in \mathcal{F}$, $x \in \mathcal{X}$,*

$$\{r, F_0, X_0\}_{\substack{r \leftarrow R,\ (F,e) \leftarrow \mathsf{Gb}(f;r), \\ (F_0, \pi_{\mathsf{En}}) \leftarrow \mathsf{Rerand}(F),\ X_0 = \mathsf{En}(\pi_{\mathsf{En}}(e), x)}} \overset{c}{\approx} \{r, F_1, X_1\}_{\substack{r \leftarrow R, (F_1, e_1) \leftarrow \mathsf{Gb}(f), \\ X_1 = \mathsf{En}(e_1, x)}}$$

where R is the space of random tapes for Gb. (Note that (F_1, e_1) is generated using fresh randomness independent of r.)

Note that Rerand-privacy and *correctness* of garbling schemes together imply that the rerandomized garbling F_0 produced by Rerand is correct – i.e., for any input x, and (F, e) produced by $\mathsf{Gb}(f)$, for $(F_0, \pi_{\mathsf{En}}) \leftarrow \mathsf{Rerand}(F)$, it must be the case that $\mathsf{Ev}(F_0, \mathsf{En}(\pi_{\mathsf{En}}(e), x)) = f(x)$ (except possibly with negligible probability). Indeed, otherwise it would be easy to distinguish this from a fresh garbling based on the outputs of garbled evaluation. Note also that Rerand does not get f as input. Therefore, it cannot operate by ignoring the prior garbling F and simply generating a fresh garbling as F'.

Definition 8 can be applied to a projective encoding as well by simply requiring that the input encoding $X' = (L_1^{'}, \cdots, L_m^{'}) = \mathsf{En}(\pi_{\mathsf{En}}(e), x)$ is projective. Formally,

Definition 9. *A* ***Projective Rerandomizable Garbling Scheme*** *is a tuple* $\mathsf{GS}' = (\mathsf{Gb}, \mathsf{Rerand}, \mathsf{En}, \mathsf{Ev})$ *where,* $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev})$ *is a projective garbling scheme (Definition 4) for a family* $\mathcal{F}$ *of functions with input domain* $\{0,1\}^m$*, and* Rerand *is a PPT algorithm as in Definition 8 that satisfies the following:*

π_{En} *produced by* Rerand *is in the form of encoding transformations* $\{\sigma_i\}_{i\in[m]}$ *such that* $\forall x \in \{0,1\}^m$*,* $\forall e \in \mathcal{E}$*,* $\mathsf{En}(e, x) = (L_1, \cdots, L_m)$ *and* $\mathsf{En}(\pi_{\mathsf{En}}(e), x) = (L_1^{'}, \cdots, L_m^{'})$*, such that* $\sigma_i(L_i) = L_i^{'}$*.*

Looking ahead, we point out that for the construction of the SCALES protocol, a slightly relaxed notion of projective RGS suffices. In this relaxed version we allow for encoding transformations of the form $\{\sigma_i^b\}_{i\in[m],b\in\{0,1\}}$ where a different transformation may be applied to the labels L_i^0 and L_i^1 to obtain their rerandomized counterparts. But we omit this for the sake of simplicity.

4.1 Strong Key and Message Homomorphic Encryption

Homomorphic encryption schemes allow the execution of mathematical operations over the plaintexts within the encrypted domain. In this work we are interested in schemes that support transformations on both the secret key and the plaintext domains within a ciphertext, resulting in a ciphertext that looks "fresh". We refer to such a scheme as a Key-and-Message Homomorphic Encryption scheme (KMHE). We abstract KMHE as a private key encryption primitive $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$,[9] that is amplified with an additional Eval algorithm. This algorithm applies two homomorphic (potentially distinct and private) transformations on a ciphertext, one on the secret key and one on the plaintext.

Definition 10. *A* ***key-and-message homomorphic encryption scheme*** *is a set of PPT algorithms* $\mathsf{KMH} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *defined on domains of (private) keys, messages and ciphertexts* $\mathcal{K}, \mathcal{M}, \mathcal{C}$*, a key transformation family* $\mathcal{F}_{key}$*, and a message transformation family* $\mathcal{F}_{msg}$ *(all indexed by an implicit security parameter* κ*) such that the following conditions hold:*

- ***Correctness****:* $\forall m \in \mathcal{M}, k \in \mathcal{K}$*,*

$$\Pr[k \leftarrow \mathsf{Gen}(1^\kappa); \mathsf{Dec}(k, \mathsf{Enc}(k, m)) = m] = 1$$

[9] For simplicity we define KMHE as a private key primitive (where encryption is carried out using the secret key). Nevertheless, the definition can be naturally extended to a public key setting as well.

– ***KMH Correctness:*** *$\forall m \in \mathcal{M}, k \in \mathcal{K}$, $f \in \mathcal{F}_{key}$, $g \in \mathcal{F}_{msg}$, $r_1, r_2 \in R$, $\exists r' \in R$,*

$$\mathsf{Eval}(\mathsf{Enc}(k, m; r_1), f, g; r_2) = \mathsf{Enc}(f(k), g(m); r')$$

where R is the space of random tapes for Enc and Eval.

– ***CPA Security:*** *$\forall$ PPT adversary $\mathcal{A}$, the advantage $\Pr[b' = b] \leq \frac{1}{2} + \nu(\kappa)$ for a negligible function ν in the following experiment (κ being an implicit input to C and $\mathcal{A}$):*
  1. *C samples a uniform random bit $b \leftarrow \{0, 1\}$.*
  2. *For as many times as $\mathcal{A}$ wants:*
     - *$\mathcal{A}$ produces arbitrary $m_0, m_1 \in \mathcal{M}$ and sends them to C.*
     - *C samples a key $k \leftarrow \mathsf{Gen}(1^\kappa)$ and sends $c_b = \mathsf{Enc}(k, m_b)$ to $\mathcal{A}$.*
  3. *$\mathcal{A}$ outputs b'.*

– ***Key Privacy:*** *$\forall k, k' \leftarrow \mathsf{Gen}(1^\kappa)$, $f \in \mathcal{F}_{key}$,*

$$\{k, f(k)\} \overset{s}{\approx} \{k, k'\}$$

Looking ahead, we use KMHE as a primitive along with RGS in the construction for *incremental Decomposable Randomized Encodings* in Section 5.

Next, we define a new object, a *strong* Key-and-Message Homomorphic Encryption scheme (strong KMHE), that has an additional security property, KMH privacy, that is required for rerandomizable garbling. We use strong KMHE as a building block in our construction for rerandomizable garbled circuits (Section 4.3).

Definition 11. *A* ***strong key-and-message homomorphic encryption*** *scheme (strong KMHE) is the set of PPT algorithms $\mathsf{KMH} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ defined on domains of (private) keys, messages and ciphertexts $\mathcal{K}, \mathcal{M}, \mathcal{C}$, a key transformation family $\mathcal{F}_{key}$, and a message transformation family $\mathcal{F}_{msg}$ (all indexed by an implicit security parameter κ) such that KMH is a KMHE scheme as in Definition 10 and the following additional condition holds:*

– ***KMH Privacy:*** *$\forall$ PPT adversary $\mathcal{A}$, the advantage $\Pr[b' = b] \leq \frac{1}{2} + \nu(\kappa)$ for a negligible function ν in the following experiment (κ being an implicit input to C and $\mathcal{A}$):*
  1. *C samples a uniform random bit $b \leftarrow \{0, 1\}$, keys $k_0, k_1, k' \leftarrow \mathsf{Gen}(1^\kappa)$, and $f \leftarrow \mathcal{F}_{key}$. It sends $(k_0, k_1, f(k_1))$ to $\mathcal{A}$.*
  2. *For as many times as $\mathcal{A}$ wants:*
     - *$\mathcal{A}$ produces arbitrary $m, m' \in \mathcal{M}$ and $g \in \mathcal{F}_{msg}$, and computes $c \leftarrow \mathsf{Enc}(k_0, m)$. It sends (c, g, m') to C.*
     - *C sends c_b to $\mathcal{A}$, where $c_0 \leftarrow \mathsf{Eval}(c, f, g)$ and $c_1 \leftarrow \mathsf{Enc}(k', m')$.*
  3. *$\mathcal{A}$ outputs b'.*

We would like to stress here that we do not require the scheme to be *fully* homomorphic, but only homomorphic with respect to certain (affine) function families. We prove that the [BHHO08] scheme satisfies strong KMHE. The details can be found in Appendix A. The [BHHO08] encryption scheme is based on the

DDH hardness assumption. We follow the construction in [GHV10] and restrict the key space $\mathcal{K}$ to all binary strings of length κ with $\frac{\kappa}{2}$ 0's and the rest 1's. In order to use this scheme for garbling, we require that $\mathcal{M} = \mathcal{K}$, and so we restrict the message space accordingly as well. The function family $\mathcal{F}_{key}$ for key domain transformations contains all permutations over κ-bit positions: $\sigma : \{0,1\}^\kappa \to \{0,1\}^\kappa$ over the sub-domain of balanced strings. Therefore, *key privacy* is maintained since $\forall k, k' \leftarrow \mathsf{Gen}(1^\kappa)$, $f \in \mathcal{F}_{key}$, the distributions $\{k, f(k)\}$ and $\{k, k'\}$ are exactly identical. [BHHO08] also supports homomorphic operations on the key and message domains in a way that *KMH privacy* is preserved. Since a scheme satisfying Definition 11 also satisfies Definition 10, to avoid overloaded notations, we instantiate both strong KMHE and KMHE in the same way.

4.2 A Gap in the proof of [GHV10]

Strong KMHE is implicit in the rerandomizable SFE protocol of [GHV10]. We outline a gap in the proof of [GHV10] in Appendix B. Informally, secure rerandomizing requires that any PPT distinguisher, given all the randomness used for a *prior* garbling M, cannot distinguish between a garbling that is rerandomized from M and a freshly created garbling M'. [GHV10] instantiated rerandomizable garbled circuits using the encryption scheme from [BHHO08] and argues that it is rerandomizable by reductions to the semantic security and key leakage resilience properties of this scheme (the latter property has been proven in [NS09]). This latter property allows semantic security even when the distinguisher is given some information about the secret key. (This is required for showing that privacy is preserved in a rerandomized GC even given *leakage* in the form of the two labels (k_0, k_1) of the *prior* GC and a transformed active label $f(k_b)$ of the RGC.)

However, such a security argument applies only to indistinguishability of two ciphertexts both encrypted under the same (transformed) key. In particular, it does not rule out adversary's ability to identify if a ciphertext was encrypted using a key obtained by transforming a known key, or from a fresh key. This allows distinguishing between a freshly garbled and a rerandomized GC.

We handle this security gap by strengthening the security definition of the underlying encryption scheme. Specifically, in our abstraction of strong KMHE, a *KMH privacy* property explicitly requires that a ciphertext computed under a fresh key be indistinguishable from a ciphertext acquired after homomorphic transformations that corresponds to a transformed key. Another security property, denoted by *key privacy*, requires that the distribution of transformed keys in the clear is indistinguishable from that of freshly sampled keys.

4.3 Constructing Rerandomizable Garbled Circuits

In this section we present a construction for rerandomizable garbled circuits. By GC rerandomization we mean a procedure that takes only the GC for a circuit C and generates another GC for the same circuit, so that the latter is indistinguishable from a freshly garbled circuit, even given input labels for one set of inputs, and all the randomness used to generate the original GC that the rerandomized GC was derived from.

We describe a GC rerandomization procedure that is based on the construction of [GHV10] with the difference that the underlying encryption scheme is a strong KMHE scheme $\mathsf{KMH} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$, as specified in Definition 11 with additional structural properties.

Sharable KMHE (Definition 12) is a strong KMHE that is modified to facilitate garbling and rerandomizing circuits. Like most traditional garbling, the RGC construction we employ also garbles a circuit gate-by-gate. Each gate is garbled by designating labels for both values 0 and 1 of each input and out wire of the gate. Then a set of ciphertexts are created by encrypting an output label using one input label from each wire, according to the truth table of the gate functionality. In our construction, these ciphertexts are created using strong KMHE. Note that since circuit intermediate wires can be both input and output wires of gates, it becomes necessary that the key space $\mathcal{K}$ of strong KMHE be a subset of the message space $\mathcal{M}$. Further, we apply the same operation to all ciphertexts with the same wire label for rerandomizing. So it is also necessary that $\mathcal{F}_{key}$ be contained in $\mathcal{F}_{msg}$.

Traditional garbling of 2-input gates employs double encryption of the output label using both input labels, creating one ciphertext. However, this would complicate matters if used as it is for RGCs since it requires the ciphertext space to also be within the message space and the domains of $\mathcal{F}_{key}$ and $\mathcal{F}_{msg}$ would need to be readjusted. Therefore, we garble by creating a 2-out-of-2 secret sharing of the output label and then encrypting each share with one input label as the key. This makes it necessary to have efficient sharing and reconstruction algorithm Share and Recon. Share is a randomized algorithm that takes as input an element from $\mathcal{K}$ and outputs two elements from $\mathcal{M}$ such that no element in isolation reveals the input. Recon is deterministic and can derive the input to Share given both shares.

In order to be compatible with rerandomizing, we require two additional properties. Firstly, we need that the shares can be rerandomized. That is, given a pair of shares s_0 and s_1 of a message k, there exist functions h and $\overline{h}$ such that $h(s_0)$ and $\overline{h}(s_1)$ are also a sharing of k. These functions must come from a domain $\mathcal{F}^*_{msg}$ such that when h is picked at random, applying h and $\overline{h}$ induces a fresh sharing of k. Since rerandomizing requires that operations are performed on ciphertexts, it becomes necessary that these operations are homomorphically applied $\mathcal{F}^*_{msg} \subseteq \mathcal{F}_{msg}$.

Secondly, note that when applying function $\sigma \in \mathcal{F}_{key}$ to the message space, it needs to be applied to shares of the label and not the label itself. Therefore, we need the additional property that any such σ applied to the shares translates to it being applied to the reconstructed value as well. Combining the two requirements, Definition 12 states that a distribution containing a sharing of a key k and, for any $\sigma \in \mathcal{F}_{key}$, the sharing of $\sigma(k)$ be identically distributed to one containing a sharing of k and a pair containing σ and h applied to one share and σ and $\overline{h}$ applied to the other.

Definition 12. *A **sharable key-and-message homomorphic encryption scheme** is a set of PPT algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval}, \mathsf{Share}, \mathsf{Recon})$ *where* $\mathsf{KMH} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *is a strong KMHE scheme as in Definition 11 for domains of (private) keys* $\mathcal{K}$*, messages* $\mathcal{M}$ *and ciphertexts* $\mathcal{C}$*, a key transformation family* $\mathcal{F}_{key}$*, and a message transformation family* $\mathcal{F}_{msg}$ *with the additional property that* $\mathcal{K} \subseteq \mathcal{M}$ *and* $\mathcal{F}_{key} \subseteq \mathcal{F}_{msg}$*.*

The scheme has two additional PPT functions (1) $([k]_0, [k]_1) \leftarrow \mathsf{Share}(k)$ *that outputs two random shares* $[k]_0, [k]_1 \in \mathcal{M}$ *of a key* $k \in \mathcal{K}$*. (2)* $k \leftarrow \mathsf{Recon}([k]_0, [k]_1)$ *that reconstructs the label* k *from its shares. Further there exists* $\mathcal{F}^*_{msg} \subseteq \mathcal{F}_{msg}$ *such that,* $\forall \sigma \in \mathcal{F}_{key}$*,* $\forall h \in \mathcal{F}^*_{msg}, \exists \overline{h} \in \mathcal{F}^*_{msg}$ *s.t.* $\forall k \in \mathcal{K}$*,*

$$\big\{[k]_0, [k]_1, [\sigma(k)]_0, [\sigma(k)]_1\big\}_{\substack{([k]_0,[k]_1)\leftarrow\mathsf{Share}(k);\\ ([\sigma(k)]_0,[\sigma(k)]_1)\leftarrow\mathsf{Share}(\sigma(k))}}$$
$$\equiv\big\{[k]_0, [k]_1, h(\sigma([k]_0)), \overline{h}(\sigma([k]_1))\big\}_{([k]_0,[k]_1)\leftarrow\mathsf{Share}(k)}$$

Construction 1. *We denote by* $\mathsf{GS} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{Rerand})$ *a rerandomizable garbling scheme where all the algorithms are instantiated with a sharable KMHE (Definition 12) scheme* KMH *as the underlying encryption scheme.*

1. *The garbling algorithm* $\mathsf{Gb}(C, 1^\kappa)$*:*
   - *For every wire* $w_i \in \mathcal{W} - O$*, sample labels* $L^0_{w_i}, L^1_{w_i} \leftarrow \mathsf{KMH.Gen}(1^\kappa)$*.*
   - *For every output wire* $w_i \in O$*, use the same labels* $\mathsf{L}_0, \mathsf{L}_1 \in \mathcal{K}$ *across all output wires. These are publicly known.*
   - *For every gate* $g_i = (w_\ell, w_r, w_i, op) \in \mathcal{G}$*, for each of the 4 rows, let* $([L^b_{w_i}]_0, [L^b_{w_i}]_1) \leftarrow \mathsf{KMH.Share}(L^b_{w_i})$ *be the shares of one of* g_i*'s output labels for* $b \in \{0, 1\}$ *and* π *be a permutation on four positions. Then the garbling of gate* g_i *can be defined as:*

$$G_i = \begin{bmatrix} \pi[0,0] : \mathsf{KMH.Enc}(L^0_{w_\ell}, [L^{op(0,0)}_{w_i}]_0), \mathsf{KMH.Enc}(L^0_{w_r}, [L^{op(0,0)}_{w_i}]_1) \\ \pi[0,1] : \mathsf{KMH.Enc}(L^0_{w_\ell}, [L^{op(0,1)}_{w_i}]_0), \mathsf{KMH.Enc}(L^1_{w_r}, [L^{op(0,1)}_{w_i}]_1) \\ \pi[1,0] : \mathsf{KMH.Enc}(L^1_{w_\ell}, [L^{op(1,0)}_{w_i}]_0), \mathsf{KMH.Enc}(L^0_{w_r}, [L^{op(1,0)}_{w_i}]_1) \\ \pi[1,1] : \mathsf{KMH.Enc}(L^1_{w_\ell}, [L^{op(1,1)}_{w_i}]_0), \mathsf{KMH.Enc}(L^1_{w_r}, [L^{op(1,1)}_{w_i}]_1) \end{bmatrix}$$

   - *Output* $\hat{\mathcal{C}} = ((G_1, \cdots, G_q), (\mathsf{L}_0, \mathsf{L}_1))$ *and* $\mathcal{L} = \{L^0_{w_i}, L^1_{w_i}\}_{w_i \in I}$*.*
2. *The encoding algorithm* $\mathsf{En}(\mathcal{L}, x)$ *gets a set of input labels* $\mathcal{L}$ *and the function input* $x = (x_1, \cdots, x_m)$ *and outputs* $\mathcal{I} = \{L^{x_i}_{w_i}\}_{w_i \in I}$*.*
3. *The evaluation algorithm* $\mathsf{Ev}(\hat{\mathcal{C}}, \mathcal{I})$ *:*
   The algorithm works gate by gate, by decrypting each row in the garbled gate.[10] *The resulting plaintexts are combined to the output label using* $\mathsf{KMH.Recon}$*. Evaluating a gate lets us derive one label for a wire in the circuit. Following the terminology of [LP09], this label is termed the active label of that wire. Such a label is also derived for each output wire of the circuit and this belongs in the set* $(\mathsf{L}_0, \mathsf{L}_1)$ *and can be mapped to output values 0 or 1. This set of labels yields the function's output* $f(x)$*.*

[10] We assume that the evaluator identifies the valid output label by adding a fixed suffix to the plaintext as suggested originally in [LP09].

4. *The rerandomizing algorithm* $\mathsf{Rerand}(\hat{\mathcal{C}})$ *:*
   - *For all wires* $w_i \in \mathcal{W} - O$*, sample* $\sigma_i \leftarrow \mathcal{F}_{key}$*.*
   - *For all output wires* $w_i \in O$*, let* $\sigma_i = \mathsf{I}$ *be the identity function.*
   - *For all gates* $g_i \in \mathcal{G}$*, let* $(\sigma_\ell, \sigma_r, \sigma_i)$ *correspond to the wires* (w_ℓ, w_r, w_i)*. Let* π_i *be a permutation on four elements. Sample* $h_0, h_1, h_2, h_3 \leftarrow \mathcal{F}^*_{msg}$ *and derive* $\overline{h_0}, \overline{h_1}, \overline{h_2}, \overline{h_3} \in \mathcal{F}^*_{msg}$*. In order to rerandomize* G_i *into* G^*_i*, the following is carried out:*

$$G'_i = \begin{bmatrix} \pi_i[0,0] : \mathsf{KMH.Eval}(c_{0,0}, \sigma_\ell, \sigma_i), \mathsf{KMH.Eval}(c_{0,1}, \sigma_r, \sigma_i) \\ \pi_i[0,1] : \mathsf{KMH.Eval}(c_{1,0}, \sigma_\ell, \sigma_i), \mathsf{KMH.Eval}(c_{1,1}, \sigma_r, \sigma_i) \\ \pi_i[1,0] : \mathsf{KMH.Eval}(c_{2,0}, \sigma_\ell, \sigma_i), \mathsf{KMH.Eval}(c_{2,1}, \sigma_r, \sigma_i) \\ \pi_i[1,1] : \mathsf{KMH.Eval}(c_{3,0}, \sigma_\ell, \sigma_i), \mathsf{KMH.Eval}(c_{3,1}, \sigma_r, \sigma_i) \end{bmatrix} \text{ where } G_i = \begin{bmatrix} c_{0,0}, c_{0,1} \\ c_{1,0}, c_{1,1} \\ c_{2,0}, c_{2,1} \\ c_{3,0}, c_{3,1} \end{bmatrix}$$

$$G^*_i = \begin{bmatrix} \mathsf{KMH.Eval}(c'_{0,0}, \mathsf{I}, h_0), \mathsf{KMH.Eval}(c'_{0,1}, \mathsf{I}, \overline{h_0}) \\ \mathsf{KMH.Eval}(c'_{1,0}, \mathsf{I}, h_1), \mathsf{KMH.Eval}(c'_{1,1}, \mathsf{I}, \overline{h_1}) \\ \mathsf{KMH.Eval}(c'_{2,0}, \mathsf{I}, h_2), \mathsf{KMH.Eval}(c'_{2,1}, \mathsf{I}, \overline{h_2}) \\ \mathsf{KMH.Eval}(c'_{3,0}, \mathsf{I}, h_3), \mathsf{KMH.Eval}(c'_{3,1}, \mathsf{I}, \overline{h_3}) \end{bmatrix} \text{ where } G'_i = \begin{bmatrix} c'_{0,0}, c'_{0,1} \\ c'_{1,0}, c'_{1,1} \\ c'_{2,0}, c'_{2,1} \\ c'_{3,0}, c'_{3,1} \end{bmatrix}$$

   - *Output* $\hat{\mathcal{C}}' = ((G^*_1, \cdots, G^*_q), (\mathsf{L}_0, \mathsf{L}_1))$ *and* $\Pi = \{\sigma_i\}_{w_i \in I}$*.*

The function $\mathsf{Rerand}(\cdot)$ has computational complexity $O(|C|)$ and the size of its output is $O(|C| \cdot \kappa)$ where κ is a security parameter.

Theorem 1. *Let* KMH *be a sharable KMHE scheme (Definition 12). Then* $\mathsf{GS} = (\mathsf{Gb}, \mathsf{Rerand}, \mathsf{En}, \mathsf{Ev})$ *(Construction 1) is an RGS with projective encoding (Definition 9).*

Proof outline. Correctness of GS is implied by the *Correctness* of KMH and the *Correctness* of the underlying garbling scheme. *Privacy* is implied by the CPA security of the encryption scheme KMH.

It remains to argue that GS preserves Rerand-*privacy*. Consider an intermediate hybrid game where, along with a prior GC and all its labels, a new GC along with active input labels are given to the distinguisher. This new GC is defined such that the active labels are generated by applying transformations $\sigma_i \leftarrow \mathcal{F}_{key}$ on each active label from the prior GC. Nevertheless, the inactive labels are still freshly sampled as in the prior garbling. Fixing these labels, the new GC is constructed as a fresh garbling. This game is indistinguishable (in fact, these distributions are statistically close), from the case where the new GC and all its labels are freshly created, via a reduction to the *Key Privacy* property of KMH. On the other hand, this game is also indistinguishable from the case where the new GC is a rerandomized garbling of the prior GC. This argument is reduced to the *KMH Privacy* property of KMH. We conclude that Rerand-*privacy* is preserved in GS.

Proof. The instantiation $\mathsf{GS} = (\mathsf{Gb}, \mathsf{Rerand}, \mathsf{En}, \mathsf{Ev})$ preserves *Correctness* by definition of the garbled circuits construction, and also from the *Correctness* of KMH (Definition 11).

Claim 1. $\mathsf{GS} = (\mathsf{Gb}, \mathsf{Rerand}, \mathsf{En}, \mathsf{Ev})$ *satisfies Privacy.*

Proof. Privacy in Garbling Schemes requires that for two functions f_0 and f_1 such that $\phi(f_0) = \phi(f_1)$, and inputs x_0 and x_1 such that $f_0(x_0) = f_1(x_1)$,

$$\{F_0, X_0\}_{\substack{(F_0,e_0)\leftarrow \mathsf{Gb}(f_0,1^\kappa);\\ \mathsf{En}(e_0,x_0)=X_0}} \overset{c}{\approx} \{F_1, X_1\}_{\substack{(F_1,e_1)\leftarrow \mathsf{Gb}(f_1,1^\kappa);\\ \mathsf{En}(e_1,x_1)=X_1}}$$

In our instantiation, for a function $f_b, b \in \{0,1\}$, F_b is the GC $\hat{\mathcal{C}}_b$ and X_b is the set of input labels corresponding to x_b: the set $\mathcal{I}^b$. Letting C_b be the circuit for f_b, $\phi(f_b)$ reveals the circuit topology, that is, all of C_b except the operation *op* for each gate $g_i \in \mathcal{G}$. We need to show that $\{\hat{\mathcal{C}}_0, \mathcal{I}^0\} \overset{c}{\approx} \{\hat{\mathcal{C}}_1, \mathcal{I}^1\}$.

This is done by first considering, for a function f and input x, the subroutine $(\hat{\mathcal{C}}', \mathcal{I}') \leftarrow \mathsf{Sim}^{GC}(f(x))$. This operates as in [LP09] and creates a garbled circuit $\hat{\mathcal{C}}'$ in which each garbled gate has ciphertexts that encrypt only the *active label*, i.e. labels that lead to the circuit evaluating to $f(x)$. The set $\mathcal{I}'$ is the corresponding set of active input labels.

In Definition 11, the KMH Privacy definition implies CPA security. Therefore the proof of the following lemma follows from that in [LP09]:

Lemma 1. *Assuming KMH Privacy (Definition 11) holds for* $\mathsf{KMH} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$*, then for any PPT adversary* A*,* $\forall x \in \{0,1\}^m$*,*

$$\{\hat{\mathcal{C}}', \mathcal{I}'\}_{(\hat{\mathcal{C}}',\mathcal{I}')\leftarrow \mathsf{Sim}^{GC}(f(x))} \overset{c}{\approx} \{\hat{\mathcal{C}}, \mathcal{I}\}_{(\hat{\mathcal{C}},\mathcal{L})\leftarrow \mathsf{Gb}(C,1^\kappa),\mathcal{I}=\mathsf{En}(\mathcal{L},x)}$$

In order to prove that GS satisfies RGS *privacy*, consider a set of hybrids where,

- $H_0 = \{\hat{\mathcal{C}}_0, \mathcal{I}^0\}_{\substack{(\hat{\mathcal{C}}_0,\mathcal{L}_0)\leftarrow \mathsf{Gb}(C_0,1^\kappa);\\ \mathcal{I}^0=\mathsf{En}(\mathcal{L}_0,x_0)}}$, the distribution using f_0 and x_0.
- $H' = \{\hat{\mathcal{C}}', \mathcal{I}'\}_{(\hat{\mathcal{C}}',\mathcal{I}')\leftarrow \mathsf{Sim}^{GC}(f_0(x_0))}$ is an intermediate hybrid.
- $H_1 = \{\hat{\mathcal{C}}_1, \mathcal{I}^1\}_{\substack{(\hat{\mathcal{C}}_1,\mathcal{L}_1)\leftarrow \mathsf{Gb}(C_1,1^\kappa)\\ \mathcal{I}^1=\mathsf{En}(\mathcal{L}_1,x_1)}}$, the distribution using f_1 and x_1.

Consider for the sake of contradiction that there exists a PPT adversary Adv that can distinguish between the distributions H_0 and H_1 with non-negligible advantage ϵ. Then it must hold that it can distinguish between either H_0 and H', or H_1 and H' with non-negligible advantage $> \frac{\epsilon}{2}$. However, if Adv can distinguish between H_0 and H', it can be used as a subroutine for a PPT adversary Adv' that can distinguish between the distributions from Lemma 1. Adv' works by receiving the set $(\hat{\mathcal{C}}, \mathcal{I})$ from the challenger and sending it to Adv. It then outputs the same bit as Adv. Therefore, Adv' has advantage $\frac{\epsilon}{2}$. However, since Lemma 1 holds, there can exist no such Adv' and therefore Adv can't distinguish these hybrids. We can similarly argue for H_1 and H' since $f_0(x_0) = f_1(x_1)$. Therefore, since Adv can distinguish between neither pair of hybrids, it must hold that $\{\hat{\mathcal{C}}_0, \mathcal{I}^0\} \overset{c}{\approx} \{\hat{\mathcal{C}}_1, \mathcal{I}^1\}$. Therefore, GS satisfies RGS privacy. □

Claim 2. $\mathsf{GS} = (\mathsf{Gb}, \mathsf{Rerand}, \mathsf{En}, \mathsf{Ev})$ *satisfies* Rerand*-Privacy.*

Proof. The Rerand-privacy property requires that for every function f, input $x \in \{0,1\}^m$, and randomness $r \in R$ used for garbling, letting

$$\{r, F_0, X_0\}_{\substack{(F,e)\leftarrow \mathsf{Gb}(f;r);\\ (F_0,e_0)\leftarrow \mathsf{Gb}(f,1^\kappa);\\ \mathsf{En}(e_0,x)=X_0}} \overset{c}{\approx} \{r, F_1, X_1\}_{\substack{(F,e)\leftarrow \mathsf{Gb}(f;r);\\ (F_1,\pi_{\mathsf{En}})\leftarrow \mathsf{Rerand}(F);\\ \mathsf{En}(\pi_{\mathsf{En}}(e),x)=X_1}}$$

In the context of our instantiation, letting r be the randomness used in the prior garbling and C be the circuit for f, we need to show that,

$$\{r, \hat{\mathcal{C}}_0, \mathcal{I}_0\}_{\substack{(\hat{\mathcal{C}},\mathcal{L})\leftarrow \mathsf{Gb}(C;r);\\ (\hat{\mathcal{C}}_0,\mathcal{L}_0)\leftarrow \mathsf{Gb}(C,1^\kappa);\\ \mathcal{I}_0=\mathsf{En}(\mathcal{L}_0,x)}} \overset{c}{\approx} \{r, \hat{\mathcal{C}}_1, \mathcal{I}_1\}_{\substack{(\hat{\mathcal{C}},\mathcal{L})\leftarrow \mathsf{Gb}(C;r);\\ (\hat{\mathcal{C}}_1,\Pi)\leftarrow \mathsf{Rerand}(\hat{\mathcal{C}});\\ \mathcal{I}_1=\mathsf{En}(\Pi(\mathcal{L}),x)}}$$

In order to show that GS satisfies Rerand-*Privacy*, consider the following hybrids:

- $J_0 = \{r, \hat{\mathcal{C}}_0, \mathcal{I}_0\}$. This corresponds to a distribution having all the randomness r for a prior (fresh) GC $\hat{\mathcal{C}}$ and input labels set $\mathcal{L}$, along with another fresh GC, $\hat{\mathcal{C}}_0$, and input labels $\mathcal{I}_0$ corresponding to x.
- $J' = \{r, \hat{\mathcal{C}}', \mathcal{I}'\}$. This distribution is the same as J_0 except $(\hat{\mathcal{C}}', \mathcal{I}') \leftarrow \mathsf{Sim}^{GC}(f(x))$. This function outputs a *simulated* garbled circuit $\hat{\mathcal{C}}'$ as in [LP09] with freshly sampled labels. Each gate in $\hat{\mathcal{C}}'$ encrypts only the active wire labels, and $\mathcal{I}'$ is a set of such active input wire labels.
- $J'' = \{r, \hat{\mathcal{C}}'', \mathcal{I}''\}$. This is generated the same way as J' except, first, a set of all active wire labels throughout the circuit are created by rerandomizing the wire labels of the GC that generated using randomness r. The set $\mathcal{I}''$ are a set of such transformed active input wire labels. Then the inactive wire labels are sampled freshly and at random. Using these, a simulated garbling $\hat{\mathcal{C}}''$ is created. The garbling procedure is carried out as in $\mathsf{Sim}^{GC}(f(x))$.
- $J_1 = \{r, \hat{\mathcal{C}}_1, \mathcal{I}_1\}$. This corresponds to the distribution where all the randomness r for a prior GC $\hat{\mathcal{C}}$ is given. Next, $(\hat{\mathcal{C}}_1, \Pi) \leftarrow \mathsf{Rerand}(\hat{\mathcal{C}})$ is created by rerandomizing the prior GC. Then, $\mathcal{I}_1 = \mathsf{En}(\Pi(\mathcal{L}), x)$ is the input labels corresponding to input x in $\hat{\mathcal{C}}_1$.

Consider for the sake of contradiction that there exists a PPT adversary Adv that can distinguish between the distributions J_0 and J_1 with non-negligible advantage ϵ. Then it must hold that it can distinguish between either J_0 and J', J' and J'', or J_1 and J'' with non-negligible advantage $> \frac{\epsilon}{3}$.

If Adv can distinguish between J_0 and J', it can be used as a subroutine in a PPT adversary Adv′ to distinguish between $\{\hat{\mathcal{C}}', \mathcal{I}'\}_{(\hat{\mathcal{C}}',\mathcal{I}')\leftarrow \mathsf{Sim}^{GC}(f_0(x_0))}$ and $\{\hat{\mathcal{C}}_0, \mathcal{I}_0\}_{\substack{(\hat{\mathcal{C}}_0,\mathcal{L}_0)\leftarrow \mathsf{Gb}(C_0,1^\kappa)\\ \mathcal{I}_0=\mathsf{En}(\mathcal{L}_0,x_0)}}$. Adv′ works by receiving a set $(\hat{\mathcal{C}}, \mathcal{I})$ from the challenger, sampling fresh randomness r and sending $(r, \hat{\mathcal{C}}, \mathcal{I})$ to Adv. Adv′ then outputs the same bit as Adv and has advantage $\frac{\epsilon}{3}$. However, since Lemma 1 holds, there can exist no such Adv′ and therefore Adv can't distinguish these hybrids.

The fact that J' and J'' are indistinguishable can be reduced to the *Key Privacy* property of KMH (Definition 11). Note that the only difference between the distributions is that for the active wire labels throughout the simulated garbled circuit, J' uses *fresh* labels $L^{'} \leftarrow \mathsf{Gen}(1^\kappa)$, whereas J'' uses labels rerandomized from those in a garbled circuit generated using randomness r. That is, for label $L \leftarrow \mathsf{Gen}(r)$ that is the active label in the prior circuit, $f \leftarrow \mathcal{F}_{key}$, each active wire label is of the form $L^{''} = f(L)$. By *key privacy*, it holds that $L^{'}$ and $L^{''}$ are drawn from distributions that are statistically close. Let their statistical distance be δ, that is negligible. Such pairs of active labels are visible to the adversary for every wire in the circuit. Therefore, letting v be the number of wires in the circuit, the total statistical distance between the distributions J' and J'' is $v \cdot \delta$, that is still negligible. Therefore, it follows that J' and J'' are also statistically close.

It remains to show that J_1 and J'' are indistinguishable:

Lemma 2. *Assuming KMH privacy (Definition 11) holds for* $\mathsf{KMH} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$*, then for any PPT adversary A, $\forall x \in \{0,1\}^m$, $r \in R$,*

$$\{r, \hat{\mathcal{C}}'', \mathcal{I}''\}_{\substack{(\hat{\mathcal{C}},\mathcal{L}) \leftarrow \mathsf{Gb}(C,r); \\ \mathcal{W}''=\{L_i^{b''}=\sigma(L_i^{b'}), L_i^{\neg b''} \leftarrow \mathsf{Gen}(1^\kappa)\}_{i\in v}; \\ (\hat{\mathcal{C}}'') \leftarrow \mathsf{Sim}^{GC}(f(x);\mathcal{W}'')}} \stackrel{c}{\approx} \{r, \hat{\mathcal{C}}_1, \mathcal{I}_1\}_{\substack{(\hat{\mathcal{C}},\mathcal{L}) \leftarrow \mathsf{Gb}(C;r); \\ (\hat{\mathcal{C}}_1, \Pi) \leftarrow \mathsf{Rerand}(\hat{\mathcal{C}}); \\ \mathcal{I}_1 = \mathsf{En}(\Pi(\mathcal{L}),x)}}$$

Proof. The proof of Lemma 2 follows from a set of hybrids $R_0, \cdots, R_v$, where v is the number of wires in the GCs $\hat{\mathcal{C}}'', \hat{\mathcal{C}}_1$. Each hybrid is of the form $R_i = (r, (\hat{\mathcal{C}}_i, \mathcal{I}^i))$ where $\mathcal{I}^i$ is the set of input labels corresponding to $\hat{\mathcal{C}}_i$ and $\hat{\mathcal{C}}_i$ is constructed as follows:

- first, using r, generate a garbled circuit $\hat{\mathcal{C}} \leftarrow \mathsf{Gb}(r)$ and let $\mathcal{W} = \{L_k^0, L_k^1\}_{k\in[v]}$ be be the set containing both the labels of all wires in $\hat{\mathcal{C}}$.
- $\forall k \in [i]$, for the wires w_k, the corresponding inactive label $L_k^{\neg b'}$ for $b \in \{0,1\}$ is chosen fresh. The active label is created by first sampling $\sigma_k \in \mathcal{F}_{key}$ and then $L_k^{b'} = \sigma_k(L_k^b)$. For all the gates that have all input / output wires as fresh, these gates are constructed freshly and as in $\mathsf{Sim}^{GC}(f(x))$. That is, all the ciphertexts encrypt shares of the active labels.
- $\forall k \in [v]\backslash[i]$, for the wires w_k, the corresponding labels $L_k^{0'}, L_k^{1'} \in \mathcal{K}$ are derived from rerandomizing those in $\hat{\mathcal{C}}$. Any gate in $\hat{\mathcal{C}}_i$ with only rerandomized wires will be a *real* gate.
- Note that there may exist gates with both fresh and rerandomized wires. A gate with both fresh input wires and one rerandomized output wire will be a *simulated* gate. A gate that has one fresh input wire and rerandomized labels for the other input wire and the output wire will also be a *simulated* gate. This is with the exception of gates with wire w_i (in hybrid R_i) as the input wires. These will be *real* gates.

Among these hybrids, $R_v = (r, \hat{\mathcal{C}}'', \mathcal{I}'')$ and $R_0 = (r, \hat{\mathcal{C}}_1, \mathcal{I}_1)$. Let A be a PPT adversary that can distinguish between R_0 and R_v with non-negligible advantage ϵ. Then there must be an index $i \in [v]$ for which A can distinguish between R_{i-1} and R_i with advantage $> \frac{\epsilon}{v}$. We show that this can be used as a subroutine for a PPT adversary A_{KMH} that can break KMH privacy. A_{KMH} would work as follows:

- the challenger C_{KMH} samples labels $L^0, L^1, L^{'} \in \mathcal{K}$, a bit $b \in \{0,1\}$, $f \leftarrow \mathcal{F}_{key}$ and computes $f(L^1)$. It gives L^0, L^1 and $f(L^1)$ to A_{KMH}.
- A_{KMH} samples the index i at random. It first samples r and generates $\hat{\mathcal{C}} \leftarrow \mathsf{Gb}(r)$ such that the labels of wire w_i are L^0 and L^1.
- let gate G_i be the garbled gate whose output wire is w_i. Let gates $\{G_n\}_{n\in[t]}$ be the set of t gates such that one of its input wires is w_i. Without loss of generality, let L^0 be the *inactive* label.
- A_{KMH} creates the labels set $\{L_n^0, L_n^1 \in \mathcal{K}\}_{n\in[t]}$ corresponding to the output wires of gates $\{G_n\}_{n\in[t]}$ in $\hat{\mathcal{C}}$. In order to make $\hat{\mathcal{C}}_i$, a set of functions $\{g_n \in \mathcal{F}_{msg}\}_{n\in[t]}$ are sampled. These are used to transform the output label shares of $\{L_n^0, L_n^1 \in \mathcal{K}\}_{n\in[t]}$. Let $[L_n^b]$ be a share of the label L_n^b. Set the tuple of $2t$ ciphertexts as $\{c_n^0 = \mathsf{Enc}(L^0, [L_n^0]), c_n^1 = \mathsf{Enc}(L^0, [L_n^1])\}_{n\in[t]}$. The other message set is $\{g_n([L_n^0]), g_n([L_n^1])\}_{n\in[t]}$.
- A_{KMH} sends functions $\{g_n\}_{n\in[t]}$, messages $\{g_n([L_n^0]), g_n([L_n^1])\}_{n\in[t]}$, and ciphertexts $\{c_n^0, c_n^1\}_{n\in[t]}$ to C_{KMH}.
- C_{KMH} creates $\{c_n^{0,b}, c_n^{1,b}\}_{n\in[t]}$ accordingly:
  Setting $f(L^1)$ as the new active label, if $b = 0$, it creates the set of ciphertexts $\{c_n^{0,0} = \mathsf{Eval}(c_n^0, f, g_n) = \mathsf{Enc}(f(L^0), [L_n^0]), c_n^{1,0} = \mathsf{Eval}(c_n^1, f, g_n) = \mathsf{Enc}(f(L^0), [L_n^1])\}_{n\in[t]}$. If $b = 1$, it samples a fresh key $L^{'}$ and creates $\{c_n^{0,1} = \mathsf{Enc}(L^{'}, [L_n^0]), c_n^{1,1} = \mathsf{Enc}(L^{'}, [L_n^1])\}_{n\in[t]}$.
- C_{KMH} sends $\{c_n^{0,b}, c_n^{1,b}\}_{n\in[t]}$, along with the corresponding public key pk to A_{KMH}. Then A_{KMH} samples for each $n \in [t]$, the function $h_n \leftarrow \mathcal{F}_{msg}^*$ and derives $\overline{h_n}$. It computes $c_n^{0,b} = \mathsf{Eval}(c_n^{0,b}, \mathsf{I}, h_n)$ and $c_n^{1,b} = \mathsf{Eval}(c_n^{1,b}, \mathsf{I}, \overline{h_n})$.
- A_{KMH} now generates $\hat{\mathcal{C}}_i, \mathcal{I}^i$ as follows:
  - $\forall k \in [i-1]$, for the wires w_k, the corresponding inactive label $L_k^{\neg b'}$ for $b \in \{0,1\}$ is chosen fresh. The active label is created by first sampling $\sigma_k \in \mathcal{F}_{key}$ and then $L_k^{b'} = \sigma_k(L_k^b)$. All the gates with these wires as inputs / outputs, are constructed freshly and all the ciphertexts encrypt shares of the active labels.
  - $\forall k \in [v]\backslash[i]$, for the wires w_k, the labels $L_k^{0'}, L_k^{1'} \in \mathcal{K}$ are derived from rerandomizing those in $\hat{\mathcal{C}}$. Any gate in $\hat{\mathcal{C}}_i$ with only rerandomized wires will be a *real* gate.
  - for wire w_i, gate G_i is constructed to be a *simulated* gate that encrypts shares of $f(L^1)$ only. Here both input wires have fresh labels.
  - For all gates $\{G_n\}_{n\in[t]}$, let the ciphertexts $\{c_n^{0,b}, c_n^{1,b}\}_{n\in[t]}$ be the encryptions of the output label share $[L_n^0]$ and $[L_n^1]$ under one input label of wire w_i, be it $f(L^0)$ or $L^{'}$. $f(L^1)$ is used as the other key for encrypting.

  Note that for the case that $b = 0$, the wire is rerandomized and the GC created corresponds to R_i. For $b = 1$, $f(L^1)$ and $L^{'}$ have no correlation, and the GC created corresponds to R_{i-1}.
- A_{KMH} sends $(r, (\hat{\mathcal{C}}_i, \mathcal{I}^i))$ to A and finally, A_{KMH} outputs whatever A outputs.

Since no such A_{KMH} can exist, therefore no such A exists. So it must hold that $J'' \overset{c}{\approx} J_1$. □

Therefore, since Adv can distinguish between neither pairs of adjacent hybrids, it must hold that GS satisfies RGS Rerand-privacy. □

Therefore, $\mathsf{GS} = (\mathsf{Gb}, \mathsf{Rerand}, \mathsf{En}, \mathsf{Ev})$ is a projective RGS satisfying Definition 9. □

5 Incremental Decomposable Randomized Encodings

In this section, we introduce a variant of Decomposable Randomized Encodings (DRE - Definition 6): an *incremental Decomposable Randomized Encoding* (iDRE). We also present a construction for an iDRE scheme based on an RGS, and a KMHE scheme (Definition 10). An iDRE is a key ingredient in realizing a secure protocol in the SCALES setting.

The goal of iDRE is to allow multiple encoders to collaborate in an encoding process while using minimal interaction. Specifically, our abstraction allows a chain of encoders to *incrementally* carry out the encoding, with each one receiving the output of the previous one. Informally, for a function f with m-bit inputs x, a chain of d encoders first each locally prepare $\{e^0_{ij}, e^1_{ij}\}_{i\in[m]}$ during an initial encoding phase (which prepares the labels and may work offline). Then, in the incremental encoding phase, the first encoder runs En to prepare an initial encoding B_1. Each subsequent encoder runs En^* which prepares B_j from B_{j-1}. Next, each input bit x_i is encoded as $Z_i = \mathsf{Combine}(\{e^{x_i}_{ij}\}_{j\in[d]}, B_d)$. The final encoding for $f(x)$ consists of $(Y, \{Z_i\}_{i\in[m]})$ where $Y \in B_d$. The formal definition below separates the encoding into PreEn and En^* to allow for better efficiency and flexibility; also $\mathsf{Combine}$ does not take all of B_d as input, but only a part of it, s_i. A basic privacy condition would require that only $f(x)$ is revealed by the final encoding; but as detailed below, we shall require a stronger privacy condition corresponding to when a subset of the encoders and input parties (combiners) are passively corrupt, privacy continues to hold.

Definition 13. ***incremental Decomposable Randomized Encoding*** *(iDRE) scheme defined for a function family $\mathcal{F}$, where each $f \in \mathcal{F}$ has domain $\{0,1\}^m$, is a tuple of polynomial time algorithms* $\mathsf{iDRE} = (\mathsf{PreEn}, \mathsf{En}, \mathsf{En}^*, \mathsf{Combine}, \mathsf{Dec})$ *for ℓ polynomial in m. Defining the following random variables as a function of $x \in \{0,1\}^m$:*

$$
\begin{aligned}
r_j &\leftarrow \{0,1\}^\ell && \forall j \in [d],\\
\{e^0_{ij}, e^1_{ij}\}_{i\in[m]} &\leftarrow \mathsf{PreEn}(j; r_j) && \forall j \in [d],\\
B_j &\leftarrow \begin{cases} \mathsf{En}(f; r_1) & \text{for } j = 1\\ \mathsf{En}^*(B_{j-1}; r_j) & \text{for } 1 < j \le d \end{cases} &&\\
\big(Y, \{s_i\}_{i\in[m]}\big) &\leftarrow B_d &&\\
Z_i &\leftarrow \mathsf{Combine}(\{e^{x_i}_{ij}\}_{j\in[d]}, s_i) && \forall i \in [m]
\end{aligned}
$$

Then the following properties need to be satisfied:

– ***Correctness***: *$\forall x \in \{0,1\}^m$, with probability 1 (over the choice of $\{r_j\}_{j\in[d]}$),*

$$\mathsf{Dec}(Y, \{Z_i\}_{i\in[m]}) = f(x).$$

– ***Privacy***: *There exists a simulator* Sim *such that $\forall x \in \{0,1\}^m$, $j^* \in [d]$ and $\mathcal{A} \subseteq [m]$,*

$$\left\{\mathsf{Sim}(f, f(x), j^*, \{x_i\}_{i\in\mathcal{A}})\right\} \overset{c}{\approx} \left\{\{r_j\}_{j\neq j^*}, B_{j^*}, \{e_{ij^*}^{x_i}\}_{i\in\mathcal{A}}, \{Z_i\}_{i\notin\mathcal{A}}\right\}$$

The privacy condition above corresponds to a semi-honest adversary who corrupts all encoders other than the one with index j^* – i.e., it learns r_j for all $j \neq j^*$, as well as the output B_{j^*}; further, for a set $\mathcal{A} \subseteq [m]$ it learns the input bits x_i as well as the label $e_{ij^*}^{x_i}$, for each $i \in \mathcal{A}$. Note that this provides the adversary with enough information to decode $f(x)$. We require that such an adversary learns nothing more about the input bits $\{x_i\}_{i\notin\mathcal{A}}$ beyond what $f(x)$ and $\{x_i\}_{i\in\mathcal{A}}$ reveals.

5.1 Realizing iDRE using RGS

In this section we outline our construction of iDRE based on a projective RGS (Definition 8) and KMHE scheme (Definition 10) which has the following design: En generates a projective garbling as well as a set of *encrypted labels*. The latter is a set of ciphertexts encrypting both labels for every input bit position within the garbling. Next, each instance of En^* takes both a garbling and its encrypted labels as inputs, and outputs a rerandomized garbling and a matching set of encrypted labels. This is achieved by modifying the encrypted plaintexts to match the labels of the new garbling by applying consistent transformations to the encrypted labels by exploiting the homomorphic properties.

Additionally, the keys under which the labels are encrypted are homomorphically refreshed by each encoder using new randomness.[11] This set of transformations is generated by the different instances of algorithm PreEn. At last, the Combine algorithm takes the final encrypted label for each input bit and all the randomness used to create the encryption key, and creates the final key that is used to decrypt the label. This label corresponds to an input label for the last GS, all given as inputs to the decoding algorithm Dec.

Notation. Let the input to the function f be $x = \{x_i\}_{i\in[m]}$. Moreover, let F_1 be the GS created by En and F_j be the rerandomized GS output by the j^{th} instance of En^*. We denote by $\mathcal{L}^j$ the set containing all the labels (corresponding to both the 0 and 1 value) for all input bit positions of F_j. Namely, $\mathcal{L}^j = \{L_{ij}^b\}_{i\in[m],b\in\{0,1\}}$, where $L_{ij}^b \in \{0,1\}^\kappa$ denotes the label used in F_j for the i^{th} input bit whose vale is $b \in \{0,1\}$. Finally, we denote the subset of *active labels* within F_j by $X^j = \{L_{ij}^{x_i}\}_{i\in[m]}$ for the input $x = \{x_i\}_{i\in[m]} \in \{0,1\}^m$.

[11] As different transformations are applied to the keys used for encrypting the different input labels, and only on the key domain, it suffices to use KMHE.

The encrypted labels set that corresponds to F_j is denoted by $\mathcal{EL}^j$ where $\mathcal{EL}^j = \{\mathsf{Enc}(K_{ij}^b, L_{ij}^b)\}_{i\in[m],b\in\{0,1\}}$. Starting with F_1, each label $L_{i1}^b \in \mathcal{L}^1$ is encrypted using a key K_{i1}^b that is chosen from a KMHE scheme. We represent by $\Pi\mathcal{K}^1 = \{K_{i1}^b\}_{i\in[m],b\in\{0,1\}}$ the set of these keys. Each subsequent $\mathcal{EL}^j$ is created from $\mathcal{EL}^{j-1}$. Namely, let $\rho_{ij}^b \in \mathcal{F}_{key}$ denote a transformation chosen to randomize the key ρ_{ij-1}^b, yielding a new transformed key ρ_{ij}^b in the key domain. Then $\Pi\mathcal{K}^j = \{\rho_{ij}^b\}_{i\in[m],b\in\{0,1\}}$ denote this set of transformations for all $j > 1$.

Another set of transformations denoted by $\pi_{\mathsf{En}} = \{\sigma_i \in \mathcal{F}_{key}\}_{i\in[m]}$ plays a different role in our construction. Namely, these transformations are applied on the plaintexts within $\mathcal{EL}^{j-1}$ with the aim of rerandomizing the input labels to match the garbling F_j. Figure 2 contains the details of the algorithms for this instantiation using a KMHE and a projective RGS. The circuit C that represents the function f is publicly available to all involved parties.

Theorem 2. *Let* $\mathsf{KMH} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *be a KMHE scheme (Definition 10) and let* $\mathsf{GS} = (\mathsf{Gb}, \mathsf{Rerand}, \mathsf{En}, \mathsf{Ev})$ *be a projective RGS (Definition 9), then Figure 2 is an iDRE (Definition 13).*

Proof Outline. The *correctness* of the construction in Figure 2 is directly implied by the *correctness* of the underlying RGS and *KMH correctness*.

In order to prove privacy holds, we define the actions of the iDRE simulator Sim. Upon receiving f, $f(x)$, a subset of the input bits $\{x_i\}_{i\in\mathcal{A}}$ and the index of the honest encoding instance j^*, it first samples $\{r_j\}_{j\neq j^*}$ and invokes the adversary on this randomness along with $\{x_i\}_{i\in\mathcal{A}}$. It then executes the pre-encoding phase honestly $\mathsf{PreEn}(j^*)$ with fresh randomness sampled internally. Next, during the encoding phase, Sim gets from the adversary a projective garbling F_{j^*-1} and a set of encrypted labels $\mathcal{EL}^{j^*-1}$. On behalf of the honest input parties, it samples arbitrary input bits $\{x'_i\}_{i\notin\mathcal{A}}$ and a new function f' such that $\phi(f) = \phi(f')$ and $f'(x') = f(x)$, and computes a fresh garbling of f', $(F_{j^*}, e) \leftarrow \mathsf{Gb}(f')$, and $X_{j^*} = \mathsf{En}(x', e)$, where $x' = \{x'_i\}_{i\notin\mathcal{A}} \cup \{x_i\}_{i\in\mathcal{A}}$. It further samples transformations for the message space to create $\mathcal{EL}^{j^*}$ and forwards $(F_{j^*}, \mathcal{EL}^{j^*})$ to the adversary. Finally, upon receiving $\{s_i\}_{i\notin\mathcal{A}}$, from the final encoder, it recreates $e_{ij}^{x'_i}$ and use them to the set $\{Z_i\}_{i\notin\mathcal{A}}$.

To prove indistinguishability between a simulation and a real executions, we define the following sequence of hybrids. H_0 is defined as the above simulation. H_1 is identical to H_0, except that f and x are used to form B_{j^*}. Next, H_2, is defined with a rerandomized garbling instead of a fresh one. Finally, H_3 is defined with the encrypted labels of both the active and inactive labels. Note that H_3 is identically distributed to the real execution.

Indistinguishability of H_0 and H_1 can be reduced to the *privacy* of RGS. The indistinguishability of H_1 and H_2 can be reduced to the Rerand-*privacy* of RGS. Lastly, the indistinguishability of H_2 and H_3 can be reduced to the *key privacy* and *CPA security* of KMHE.

Proof. The *correctness* of the construction in Figure 2 is directly implied by the *correctness* of the underlying RGS and *KMH correctness*.

iDRE construction using projective RGS

Building blocks:
Projective RGS: $\mathsf{GS} = (\mathsf{Gb}, \mathsf{Rerand}, \mathsf{En}, \mathsf{Ev})$
KMHE: $\mathsf{KMH} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ such that the set of encoding transformations of GS is a subset of the message transformation family $\mathcal{F}_{msg}$ of KMH

Function $f : \{0,1\}^m \to \{0,1\}^l$
Function input $x = (x_1, \cdots, x_m)$

function $\mathsf{PreEn}(j; r_j)$
  parse $r_j = (r'_j, r^0_{1j}, \cdots, r^0_{mj}, r^1_{1j}, \cdots, r^1_{mj})$
  if $j = 1$ **then**
    $e^b_{i1} = K^b_{i1} \leftarrow \mathsf{KMH.Gen}(1^\kappa; r^b_{ij})$ $\forall i \in [m], b \in \{0,1\}$
  else
    $e^b_{ij} = \rho^b_{ij} \leftarrow \mathcal{F}_{key}$ using r^b_{ij} $\forall i \in [m], b \in \{0,1\}$
  return $\{e^0_{ij}, e^1_{ij}\}_{i\in[m]}$

function $\mathsf{En}(f; r_1)$
  parse $r_1 = (r'_1, r^0_{11}, \cdots, r^0_{m1}, r^1_{11}, \cdots, r^1_{m1})$
  $\{K^b_{i1}\}_{i\in[m],b\in\{0,1\}} \leftarrow \mathsf{PreEn}(1; r_1)$
  $(F_1, e_1) \leftarrow \mathsf{GS.Gb}(f; r'_1)$
  $(L^b_{11}, \cdots, L^b_{m1}) \leftarrow \mathsf{GS.En}(b^m, e_1)$ $\forall b \in \{0,1\}$
  $\alpha^b_{i1} \leftarrow \mathsf{KMH.Enc}(K^b_{i1}, L^b_{i1})$ $\forall i \in [m], b \in \{0,1\}$
  return $(F_1, \{\alpha^b_{i1}\}_{i\in[m],b\in\{0,1\}})$

function $\mathsf{En}^*(B_{j-1}; r_j)$
  parse $B_{j-1} = (F_{j-1}, \{\alpha^b_{ij-1}\}_{i\in[m],b\in\{0,1\}})$
  parse $r_j = (r'_j, r^0_{1j}, \cdots, r^0_{mj}, r^1_{1j}, \cdots, r^1_{mj})$
  $\{\rho^b_{ij}\}_{i\in[m],b\in\{0,1\}} \leftarrow \mathsf{PreEn}(j; r_j)$
  $(F_j, \{\sigma_i\}_{i\in[m]}) \leftarrow \mathsf{GS.Rerand}(F_{j-1}; r'_j)$
  $\alpha^b_{ij} \leftarrow \mathsf{KMH.Eval}(\alpha^b_{ij-1}, \rho^b_{ij}, \sigma_i)$ $\forall i \in [m], b \in \{0,1\}$
  return $B_j = (F_j, \{\alpha^b_{ij}\}_{i\in[m],b\in\{0,1\}})$

function $\mathsf{Combine}(\{e^{x_i}_{ij}\}_{j\in[d]}, s_i)$
  parse $\{e^{x_i}_{ij}\}_{j\in[d]} = (K^{x_i}_{i1}, \rho^{x_i}_{i2}, \cdots, \rho^{x_i}_{id})$
  parse $s_i = (\alpha^0_{id}, \alpha^1_{id})$
  $K^{x_i}_{id} \leftarrow \rho^{x_i}_{id} \circ \cdots \circ \rho^{x_i}_{i2}(K^{x_i}_{i1})$
  return $\mathsf{KMH.Dec}(K^{x_i}_{id}, \alpha^{x_i}_i)$

function $\mathsf{Dec}(Y, \{Z_i\}_{i\in[m]})$
  return $\mathsf{GS.Ev}(Y, \{Z_i\}_{i\in[m]})$

Fig. 2 Instantiating an iDRE using a projective RGS and KMHE

Claim 3. *The construction in Figure 2 satisfies privacy of iDRE.*

Proof. In order to show that privacy holds, we define the actions of the iDRE simulator Sim. Sim knows f, the function output $f(x)$, a subset of the input bits $\{x_i\}_{i\in\mathcal{A}}$ and the index of the honest encoding instance j^*. Sim works as follows:

- It first samples $\{r_j\}_{j\neq j^*}$ and invokes the adversary with $\{r_j\}_{j\neq j^*}$ and $\{x_i\}_{i\in\mathcal{A}}$.
- During the pre-encoding phase, if $j^* = 1$, then it generates weak KMHE (Definition 10) keys for both values for each input bit, including those of the adversary, $\{e^{x_i}_{i1} = K^{x_i}_{i1} \in \mathcal{K}\}_{i\in\mathcal{A}}$. Otherwise, it samples key domain

permutations at random for the same, those of the adversary being $\{e^{x_i}_{ij^*} = \rho^{x_i}_{ij^*} \in \mathcal{F}_{key}\}_{i\in\mathcal{A}}$.

- During the encoding phase, first, for all input bit positions whose bits are unknown to Sim, it samples at random $\{x'_i\}_{i\notin\mathcal{A}}$. Let $x' = \{x'_i\}_{i\notin\mathcal{A}} \cup \{x_i\}_{i\in\mathcal{A}}$. Let $\phi(\cdot)$ be a leakage function representing the information that the garbling leaks about the function garbled. Next, it picks a function $f' \in \mathcal{F}$ such that $\phi(f) = \phi(f')$ and $f(x) = f'(x')$. Sim computes the projective garbling $(F_{j^*}, e) \leftarrow \mathsf{Gb}(f')$. It computes $X' = \mathsf{En}(x', e)$ as the set of projective labels $X' = \{L^{x_i}_{ij^*}\}_{i\in[m]}$ corresponding to the active input values.
- If $j^* = 1$, Sim uses $\{K^b_{i1}\}_{b\in\{0,1\},i\in[m]}$ that it sampled during the pre-encoding phase to create the encrypted labels set $\mathcal{EL}^1 = \{\mathsf{Enc}(K^b_{i1}, L^{x_i}_{i1})\}_{b\in\{0,1\},i\in[m]}$ using weak KMHE. Then F_1 and $\mathcal{EL}^1$ are passed to the adversary.
- If $j^* \neq 1$, Sim receives $B_{j^*-1} = (F_{j^*-1}, \mathcal{EL}^{j^*-1})$ from the adversary. Given the previous encrypted labels $\mathcal{EL}^{j^*-1}$ and knowing $\{r_j\}_{j\neq j^*}$, each old plaintext label $L^b_{ij^*-1}$ corresponding to F_{j^*-1} can be derived. Now, Sim chooses transformations $\sigma^0, \sigma^1 \in \mathcal{F}_{msg}$ such that $\sigma^0(L^0_{ij^*-1}) = L^{x_i}_{ij^*}$ and $\sigma^1(L^1_{ij^*-1}) = L^{x_i}_{ij^*}$. These are applied to the ciphertexts in $\mathcal{EL}^{j^*-1}$ in the message domain. Next, it applies $\{\rho^b_{ij^*}\}_{b\in\{0,1\},i\in[m]}$ that it sampled during the pre-encoding phase in the key domain for each corresponding ciphertext. $\mathcal{EL}^{j^*}$ is hence formed and $B_{j*} = (F_{j^*}, \mathcal{EL}^{j^*})$ is given to the adversary.
- After the encoding phase is completed, Sim receives $\{s_i\}_{i\notin\mathcal{A}} \in B_d$ from the adversary. Using each randomness r_j, it recreates all $e^{x'_i}_{ij}$ for the honest input bits it sampled. These are used in Combine and the set $\{Z_i\}_{i\notin\mathcal{A}}$ is created and given to the adversary.

Therefore, the view of the adversary that the simulator generates consists of,

$$\{r_j\}_{j\neq j^*}, \{e^{x_i}_{ij^*}\}_{i\in\mathcal{A}}, B_{j*}, \{Z_i\}_{i\notin\mathcal{A}}$$

In order to prove indistinguishability between the output of Sim and a real execution, we define the following hybrids: H_0 is the same distribution as the simulation above.
We define an intermediate H_1 to be a distribution similar to H_0, except, during the encoding phase, while creating F_{j^*} and $\mathcal{EL}^{j^*}$, the garbling $(F_{j^*}, e) \leftarrow \mathsf{Gb}(f)$ comes from the original function f and the active labels encoded are those corresponding to x, instead of x'.
A hybrid H_2 is defined the same as H_1, with the exception that F_{j^*} is a *rerandomized* garbling and the active labels encrypted in $\mathcal{EL}^{j^*}$ are rerandomized to match.
The last hybrid H_3 has the same view as the real execution. The difference between this and H_2 is is that $\mathcal{EL}^{j^*}$ now has both the active and inactive labels as the plaintexts.

Now consider by contradiction that there exists a PPT distinguisher D that can distinguish between H_0 and H_3 with non-negligible advantage ϵ. Then it must follow that it can distinguish between some pair of adjacent hybrids with advantage $\frac{\epsilon}{3}$.

If D can distinguish between H_0 and H_1, it can be used as a subroutine in the PPT distinguisher D' that breaks *privacy* of GS (Definition 3). D' works as follows:

- D' chooses two functions $f_0 = f$ and $f_1 = f'$ such that $\phi(f) = \phi(f')$. It sets the inputs as $x_0 = x$ and $X_1 = x'$ such that $f_0(x_0) = f_1(x_1)$. Then, f_0, f_1, x_0, x_1 are sent to the challenger C.
- C samples $b \in \{0, 1\}$ and sends back (F_b, X_b) that is created as $(F_b, e) \leftarrow \mathsf{Gb}(f_b)$ and $X_b = \mathsf{En}(x_b, e)$.
- D' sets F_b as F_{j^*} in the view and creates $\mathcal{EL}^{j^*}$ using X_b as in the simulation. It also generates the rest of the view exactly as in the simulation.
- D' gives this view to D and outputs whatever D outputs.

Since *privacy* holds for GS, it follows that no such D' can exist and so no such D can exist. Therefore, $H_0 \overset{c}{\approx} H_1$.

If D can distinguish between H_1 and H_2, it can be used as a subroutine in the PPT distinguisher D' that breaks Rerand-*privacy* of GS(Definition 8). D' works as follows:

- D' gives the challenger C the function f and input x.
- C first samples randomness r and uses r to create a prior garbling: $(F, e) \leftarrow \mathsf{Gb}(f, r)$. Next, it samples a bit $b \in \{0, 1\}$. If $b = 0$, it creates a rerandomized garbling $(F_0, \pi_{\mathsf{En}}) \leftarrow \mathsf{Rerand}(F)$ and $X_0 = \mathsf{En}(x_i, \pi_{\mathsf{En}}(e)$. If $b = 1$, it creates a fresh garbling $(F_1, e_1) \leftarrow \mathsf{Gb}(f)$ and $X_1 = \mathsf{En}(x_i, e_1)$. Then (r, F_b, X_b) are sent to D'.
- D' uses r to set $\{r_j\}_{j \neq j^*}$ and sets F_{j^*} as F_b. It creates $\mathcal{EL}^{j^*}$ using X_b as in the simulation. It also generates the rest of the view exactly as in the simulation and is sent to D.
- Finally, D' outputs whatever D outputs.

Since Rerand-*privacy* holds for GS, it follows that no such D' can exist and so no such D can exist. Therefore, $H_1 \overset{c}{\approx} H_2$.

Lastly, if D can distinguish between H_2 and H_3, it can be used as a subroutine in the PPT distinguisher D' that breaks *CPA security* of KMHE (Definition 10). D' works as follows:

- Letting m be the number of input bits, D' first creates $(F_{j^*}, e) \leftarrow \mathsf{Gb}(f')$ and $X_0 = \mathsf{En}(x', e)$. Let $x'' = \neg x$ be the input bits for the inactive labels. It also computes $X_1 = \mathsf{En}(x'', e)$.
- D' sends X_0 and X_1 to the challenger C.
- C samples a bit $b \in \{0, 1\}$. It samples m different keys and returns the set $\mathcal{EL}^b = \{\mathsf{Enc}(k_i, L_i)\}_{L_i \in X_b}$.
- D' samples m more keys and creates $\mathcal{EL}' = \{\mathsf{Enc}(k_i, L_i)\}_{L_i \in X_0}$. It sets $\mathcal{EL}^{j^*} = \mathcal{EL}' \cup \mathcal{EL}^b$.
- D' then completes the view as in the simulation and sends it to D.

– Finally, D' outputs whatever D outputs.

Since *CPA security* holds for KMHE, it follows that no such D' can exist and so no such D can exist. Therefore, conditioned on the fact that a fresh key (as sampled by the challenger) is statistically close to a rerandomized key (as in the real execution, after multiple key transformations), $H_2 \overset{c}{\approx} H_3$. This condition is ensured by the *key privacy* property of KMHE (Definition 10).

This concludes the proof. □

Therefore, the construction in Figure 2 satisfies Definition 13. Letting d be the total number of encoding instances, the functions $\forall j \in [d], \{e^0_{ij}, e^1_{ij}\}_{i\in[m]} \leftarrow \mathsf{PreEn}(r_j; j)$, in the instantiation, return $\Pi\mathcal{K}^j$. For $j = 1$, this is a set of keys from the key space $\mathcal{K}$ of KMH. Otherwise, it is a set of elements in $\mathcal{F}_{key}$. Each such element is a κ-bit string. So the size of each e^b_{ij} for this instantiation is κ.
□

6 Realizing SCALES

In Construction 2, we show how one can obtain a SCALES scheme from an iDRE scheme, combined with a 2-message OT protocol (with semi-honest, adaptive-receiver security), $\Pi^{\mathsf{OT}} = (\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_{out})$ (corresponding to computing the receiver's message and state, the sender's message, and the receiver computing its output) as described in Section 2.3. The construction is quite simple: Each P_i encodes x_i as $(z_i, w_i) = \mathsf{OT}_1(x_i)$ and posts z_i. The ephemeral servers play the role of the encoders in iDRE: E_j will post the encoding B_j and also, for each input party P_i, it will post $\mathsf{OT}_2(z_i, e^0_{ij}, e^1_{ij})$ on the bulletin board. Afterwards, each input party P_i reads the OT messages posted by each E_j, and using w_i, recovers $e^{x_i}_{ij}$; then it runs $\mathsf{Combine}$ and posts the result back on the bulletin board. The final output computation is done using iDRE's Dec algorithm.

Construction 2. *Let f be the function for input $x = (x_1, \cdots, x_m)$ where x_i is P_i's private input. Let $\mathsf{iDRE} = (\mathsf{PreEn}, \mathsf{En}, \mathsf{En}^*, \mathsf{Combine})$ be the iDRE (Definition 13) for f and $\Pi^{\mathsf{OT}} = (\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_{out})$ be the OT protocol as above. Then the algorithms in SCALES are instantiated as:*

– $\forall i \in [m], (z_i, w_i) \leftarrow \mathsf{InpEnc}(i, x_i; t_i)$ -
  - *output* $(z_i, w_i) \leftarrow \mathsf{OT}_1(x_i; t_i)$ *where z_i is the OT first message*
– $\forall j \in [d], \alpha_j \leftarrow \mathsf{FEnc}(\mathcal{B}_{j-1}; r_j)$ -
  - *if* $j = 1$, *compute* $B_1 = \mathsf{iDRE.En}(f; r_1)$
  - *if* $j \neq 1$, *compute* $B_j = \mathsf{iDRE.En}^*(j, B_{j-1}; r_j)$ *using* $B_{j-1} \in \mathcal{B}_{j-1}$
  - *compute* $\{e^0_{ij}, e^1_{ij}\}_{i\in[m]} = \mathsf{iDRE.PreEn}(j; r_j)$
  - *compute* $\forall i \in [m], m^{i,j}_2 \leftarrow \mathsf{OT}_2(z_i, (e^0_{ij}, e^1_{ij}))$
  - *output* $\alpha_j = \{B_j, \{m^{i,j}_2\}_{i\in[m]}\}$
– $\forall i \in [m], y_i \leftarrow \mathsf{Aggregate}(\mathcal{B}_d, w_i)$ -
  - *compute* $\forall j \in [d], e^{x_i}_{ij} \leftarrow \mathsf{OT}_{out}(w_i, m^{i,j}_2)$ *using* $m^{i,j}_2 \in \mathcal{B}_d$
  - *output* $y_i = \mathsf{iDRE.Combine}(\{e^{x_i}_{ij}\}_{j\in[n]}, s_i)$ *using* $s_i \in \mathcal{B}_d$
– $y \leftarrow \mathsf{Decode}(\mathcal{B}_d, \{y_i\}_{i\in[m]})$ -
  - *output* $f(x) = \mathsf{iDRE.Dec}(\hat{f}_0(r), \{y_i\}_{i\in[m]})$ *using* $\hat{f}_0(r) \in \mathcal{B}_d$

Complexity. We note that in this construction, each ephemeral server carries out one execution of PreEn and En^* (or En) and m executions of OT_2 (reading their inputs from the bulletin board, and posting the outputs back there); when instantiated using our iDRE construction, this translates to $O(\kappa|f|)$ computational and communication complexity for each server. More importantly, note that each input party carries out a single execution of OT_1, d instances of OT_{out}, and a single instance of $\mathsf{Combine}$, all of which are independent of the complexity of f.

Theorem 3. *Let* $\mathsf{iDRE} = (\mathsf{PreEn}, \mathsf{En}, \mathsf{En}^*, \mathsf{Combine})$ *be an iDRE (Definition 13) for the function family* $\mathcal{F}$ *where each* $f \in \mathcal{F}$ *has domain* $\{0,1\}^m$ *and let* $\Pi^{\mathsf{OT}} = (\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_{out})$ *be a 2-message OT protocol (Section 2.3) that semi-honest securely computes the 2-party OT functionality* OT *in the presence of a static-corrupted sender and an adaptively corrupted receiver. Then the protocol described in Construction 2 is a secure SCALES scheme (Definition 7).*

Proof outline. The *correctness* for Construction 2 follows from the *correctness* of the underlying iDRE and the OT protocol.

Next, for function output $f(x)$, let $\mathcal{A}_1 \subset [m]$ be the indices of the subset of input providers that are initially corrupted, $\mathcal{A}_2 \subset [m]$ be the indices of the input providers that are adaptively corrupted, and j^* be the index of the single honest encoder. These are the inputs for the SCALES simulator Sim. The description of Sim is based on the output of Sim', the iDRE simulator (Definition 13), and the OT simulators, performing different actions for the case that the sender is corrupted or not.

Within Sim, Sim_1 first runs an instance of $\mathsf{Sim}_1^{\mathsf{OT}}$ each for all input providers that are not statically corrupted. Next, it runs the iDRE simulator Sim' with inputs $f(x)$ and the input bits of the statically corrupted input providers. Using the randomness returned by this, Sim_1 can create the OT input for each corrupt encoder. These are used in OT_2 to create second OT messages. For the honest encoder, $\mathsf{Sim}_2^{\mathsf{OT}}$ is used to produce the second OT message. All this is also used to generate the complete state of the bulletin board, completing the view that Sim_1 needs to output.

Next, Sim_2 executes with the input bits of the adaptively corrupted input providers as its additional input. Here, for each adaptively corrupted input bit, $\mathsf{Sim}_3^{\mathsf{OT}}$ is executed to output a candidate randomness t that can be used to explain m_1 and m_2 generated previously in the protocol.

We prove indistinguishability between the simulation and the real executions by a sequence of four hybrids. Let H_0 be the simulated distribution as outlined above. Next, H_1 differs from H_0 by switching to real iDRE function executions as opposed to simulated executions in the prior hybrid. The indistinguishability between H_0 and H_1 is reduced to the privacy of the iDRE. Next, for all input providers that are not statically corrupted the first OT messages in H_2 are also generated as in the real execution, instead of using $\mathsf{Sim}_1^{\mathsf{OT}}$. The two hybrids are proven indistinguishable based on a reduction to the receiver OT privacy.

Finally, H_3 is the real execution. Note that the only difference between H_2 and the real execution is that all OT second messages are no longer simulated in H_3. A similar argument made here as well, reducing the indistinguishability between these two hybrids into the privacy of the honest OT sender.

Proof. The *correctness* for Construction 2 follows from the *correctness* of the underlying iDRE and the OT protocol.

Claim 4. *Construction 2 satisfies privacy of SCALES.*

In this setting, since servers do not maintain state, we only consider static corruption for the servers $E_j \in \mathcal{S}$. In contrast, we consider adaptive corruption for the input clients. Consider a semi-honest adversary Adv that statically corrupts all but one encoding servers $\mathcal{E} = \{E_j\}_{j\in[d],j\neq j^*}$ and a subset of the input providers $\mathcal{P}_1 = \{P_i\}_{i\in\mathcal{A}_1\subset[m]}$. Then, after the input providers post their first message, Adv further corrupts the set $\mathcal{P}_2 = \{P_i\}_{i\in\mathcal{A}_2\subset[m]}$. The final view of the adversary includes the function f, the output $f(x)$, and the input bits $\{x_i\}_{i\in\mathcal{A}_1\cup\mathcal{A}_2}$ of all the corrupt input providers. These are given to a SCALES simulator Sim that also gets the index j^* of the honest server and outputs a view that is required to be indistinguishable from,

$$\{\mathcal{B}_d, \{y_i\}_{i\in[m]}, \{r_j\}_{j\in[d]\setminus\{j^*\}}, \{t_i\}_{i\in\mathcal{A}_1}, \{t_i\}_{i\in\mathcal{A}_2}\}$$

We define the SCALES simulator to be a pair of PPT algorithms $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$. It uses as a subroutine the iDRE simulator Sim' (Definition 13) and an OT simulator $\mathsf{Sim}^{\mathsf{OT}} = (\mathsf{Sim}^{\mathsf{OT}}_1, \mathsf{Sim}^{\mathsf{OT}}_2, \mathsf{Sim}^{\mathsf{OT}}_3)$ (Section 2.3). The following are the inputs and outputs of these subroutines:

$$\begin{aligned}(\{r'_j\}_{j\neq j^*}, B_{j^*}, \{e^{x_i}_{ij^*}\}_{i\in\mathcal{A}_1}, \{Z_i\}_{i\notin\mathcal{A}_1}) &\leftarrow \mathsf{Sim}'(f, f(x), j^*, \{x_i\}_{i\in\mathcal{A}_1})\\ (m_1, \mathsf{state}) &\leftarrow \mathsf{Sim}^{\mathsf{OT}}_1(\cdot)\\ (m_2, \mathsf{state}') &\leftarrow \mathsf{Sim}^{\mathsf{OT}}_2(m_1, \mathsf{state})\\ t &\leftarrow \mathsf{Sim}^{\mathsf{OT}}_3(\mathsf{state}', b, e_b)\end{aligned}$$

We now define the actions of the simulator. First, we define the actions of Sim_1. It gets as input the function f and its output $f(x)$, the index j^* of the honest server, and the input bits $\{x_i\}_{i\in\mathcal{A}_1}$ of the statically corrupted input providers. It then performs the following:

- Sim_1 obtains a simulated view for the iDRE invoking $\mathsf{Sim}'(f, f(x), j^*, \{x_i\}_{i\in\mathcal{A}_1})$ to get $(\{r'_j\}_{j\neq j^*}, B_{j^*}, \{e^{x_i}_{ij^*}\}_{i\in\mathcal{A}_1}, \{Z_i\}_{i\notin\mathcal{A}_1})$.
- it sets $\{r_j\}_{j\neq j^*} = \{r'_j\}_{j\neq j^*}$.
- on behalf of all input providers $P_i \in \mathcal{P} - \mathcal{P}_1$ that are not statically corrupted, Sim_1 invokes $(m^i_1, \mathsf{state}) \leftarrow \mathsf{Sim}^{\mathsf{OT}}_1(\cdot)$.
- each statically corrupted input player generates m^i_1 using OT_1. This completes the set of all first OT messages on $\mathcal{B}_d$.
- for each corrupt encoder E_j, using r_j, Sim_1 can obtain the string inputs to the OT functionality: $\{e^0_{ij}, e^1_{ij}\}_{i\in[m]} \leftarrow \mathsf{PreEn}(j; r_j)$. With these, using OT_2, all the OT second messages $\{m^{i,j}_2\}_{i\in[m],j\neq j^*}$ for the corrupt encoders are created.

– on behalf of the honest encoder, Sim_1 invokes m instances of $(m_2^{i,j^*}, \mathsf{state}') \leftarrow \mathsf{Sim}_2^{\mathsf{OT}}(m_1^i, \mathsf{state})$. This completes the set of all second OT messages on $\mathcal{B}_d$.
– finally, for each corrupt encoder E_j, using r_j, compute B_j using En or En^* as required. This completes the view of $\mathcal{B}_d$ as required.
– for the statically corrupt input providers, using $\{e_{ij^*}^{x_i}\}_{i\in\mathcal{A}_1}$ from the iDRE simulation, all the $\{e_{ij}^{x_i}\}_{i\in\mathcal{A}_1, j\neq j^*}$ created using PreEn, and $\{s_i\}_{i\in\mathcal{A}_1}$ compute $\mathsf{Combine}$ to get each $\{Z_i\}_{i\in\mathcal{A}_1}$.
– for the initially honest input providers, $\{Z_i\}_{i\notin\mathcal{A}_1}$ is part of the iDRE simulation output. These complete the set $\{y_i\}_{i\in[m]}$.
– finally, for each statically corrupt input provider, Sim_1 computes $t_i \leftarrow \mathsf{Sim}_3^{\mathsf{OT}}(\mathsf{state}', x_i, e_{ij^*}^{x_i})$ and derives the set $\{t_i\}_{i\in\mathcal{A}_1}$.

Next, we define the actions of Sim_2. It gets as input a state variable from Sim_1 that typically contains all its input, output and randomness; and additionally, the input bits $\{x_i\}_{i\in\mathcal{A}_2}$ for the input providers that have been adaptively corrupted. Sim_2 works by simply invoking for each $P_i \in \mathcal{P}_2$, the OT simulator $t_i \leftarrow \mathsf{Sim}_3^{\mathsf{OT}}(\mathsf{state}', x_i, e_{ij}^{x_i})$ to obtain the receivers randomness that explains its OT messages. Sim_2 then outputs $\{t_i\}_{i\in\mathcal{A}_2}$.

We now prove that the view generated by Sim is indistinguishable from that in the real execution of the functions by employing the following set of hybrids:

– The initial hybrid H_0 is the same as the view output by the simulator.
– The hybrid H_1 is generated in the same way as H_0 except that instead of using the iDRE simulator subroutine Sim', $(\{r_j\}_{j\neq j^*}, B_{j^*}, \{Z_i\}_{i\in[m]})$ come from a real execution of the iDRE functions.
– The next hybrid H_2 is the same as H_1 except in H_2, for all statically corrupted input providers the OT first message comes from a real execution of OT_1 as opposed to coming from a simulation $\mathsf{Sim}_1^{\mathsf{OT}}$ subroutine.
– Next, hybrid H_3 is generated the same way as H_2 except for the fact that for the honest encoder, the second OT message is derived from an execution of OT_2 instead of from $\mathsf{Sim}_2^{\mathsf{OT}}$. As a result, all the receivers randomness in this view is also the real randomness used, as opposed to being simulated using $\mathsf{Sim}_3^{\mathsf{OT}}$. Therefore, this view is the same as in the real execution.

Consider for the sake of contradiction that there exists a PPT distinguisher D that can, with non-negligible advantage ϵ distinguish between a simulation H_0 and a view in the real execution H_3. Then it follows that there exists an index i such that D would have at least $\frac{\epsilon}{3}$ advantage in distinguishing between the adjacent hybrids H_i and H_{i-1}.

If D could distinguish between hybrids H_0 and H_1, then it can be used as a subroutine by a PPT distinguisher D' that can break iDRE privacy (Definition 13) with advantage $\frac{\epsilon}{3}$. D' would work as follows:

– The challenger has the public function f and the input x.
– D' would give the challenger an index j^* for the honest encoder, and a set of input indices $\mathcal{A}_1 \subset [m]$.

– D' receives a tuple $(\{r_j\}_{j \neq j^*}, B_{j^*}, \{Z_i\}_{i \in [m]})$ along with $\{x_i\}_{i \in \mathcal{A}_1}$, the input bits of the statically corrupted input providers.
– Using this challenge, it generates the rest of the view exactly as in the simulation and passes it on to D.
– finally, D' outputs whatever D outputs.

Note that if the challenge is a simulated view of the iDRE, then D receives a tuple distributed as in H_0. It receives a tuple from H_1 otherwise. D' has advantage $\frac{\epsilon}{3}$ in this game, which is non negligible. However, since iDRE privacy holds, no such D' could exist and so D can't distinguish the two distributions.

In order to show that $H_1 \stackrel{c}{\approx} H_2$, we define a sequence of hybrids. Let $t = m - |\mathcal{A}_1|$ be the number of input providers that are not statically corrupted. Then the hybrids $J_0, \cdots, J_t$, are each of the form where in J_i, the first i OT first messages of these honest input providers come from a real execution of OT_1 and the rest come from $\mathsf{Sim}_1^{\mathsf{OT}}$. In all the hybrids, the iDRE outputs belong to the real execution and so $J_0 = H_1$ and $J_t = H_2$. If a PPT distinguisher D that can distinguish for SCALES privacy can distinguish between H_1 and H_2, then it follows that there must exist an index $i \in [t]$ for which it can distinguish between neighbouring hybrids J_{i-1} and J_i with a non-negligible advantage $> \frac{\epsilon}{3t}$. Such a D can be used as a subroutine by a PPT distinguisher D' that can distinguish for OT receiver privacy:

– D' is given a challenge message m_1. It first chooses a position $i \in [t]$ uniformly at random.
– D' samples a function f and input $x = (x_1, \cdots, x_m)$. It locally generates the iDRE outputs as real executions of the iDRE functions.
– For all corrupt input providers $P_i \in \mathcal{P}_1$, the OT first message m_1^i is generated using OT_1.
– For the first $i-1$ honest input providers, the OT first message $m_1^{i'}$ is also generated using OT_1. For all honest input providers after the i^{th} input provider, this message is created using Sim_1. For the i^{th} input provider, the challenge m_1 is set as its OT first message.
– The rest of the view for D is completed as in the simulation and is sent to D.
– Finally, D' outputs whatever D outputs.

However, since no such D' can exist, no such D can distinguish for any index between J_i and J_{i-1}. Therefore, $H_2 \stackrel{c}{\approx} H_1$.

In order to show that $H_2 \stackrel{c}{\approx} H_3$, we define a sequence of hybrids. Let $t = m$ be the number of OT second messages that are simulated on behalf of the honest encoder j^*. Then the hybrids $J_0, \cdots, J_t$, are each of the form where in J_i, the first i OT second messages come from a real execution of OT_2 and the rest come from $\mathsf{Sim}_2^{\mathsf{OT}}$. In all the hybrids, the iDRE outputs belong to the real execution and all the OT first messages also belong to the real execution of OT_1. So $J_0 = H_2$ and $J_t = H_3$. If a PPT distinguisher D that can distinguish for SCALES privacy can distinguish between H_2 and H_3, then it follows that there must exist an index

$i \in [t]$ for which it can distinguish between neighbouring hybrids J_{i-1} and J_i with a non-negligible advantage $> \frac{\epsilon}{3t}$. Such a D can be used as a subroutine by a PPT distinguisher D' that can distinguish for OT sender privacy:

- D' is given a challenge message m_2. It first chooses a position $i \in [m]$ uniformly at random.
- D' samples a function f and input $x = (x_1, \cdots, x_m)$. It locally generates the iDRE outputs as real executions of the iDRE functions. Then it generates all the OT first messages using OT_1.
- For all corrupt encoders $E_j, j \neq j^*$, each OT second message $m_2^{i,j}$ is generated using OT_2.
- For the honest encoder, the first $i-1$ OT second messages m_2^{i',j^*} are also generated using OT_2. Corresponding to all input providers after the i^{th} input provider, this message is created using Sim_2. For the i^{th} input provider, the challenge m_2 is set as its OT second message.
- The rest of the view for D is completed as in the real execution and this is sent to D.
- Finally, D' outputs whatever D outputs.

However, since no such D' can exist, no such D can distinguish for any index between J_i and J_{i-1}. Therefore, $H_2 \overset{c}{\approx} H_3$.

Therefore, it follows that since D can distinguish between neither H_0 and H_1, H_1 and H_2, and nor H_2 and H_3 with non-negligible advantage, D can't distinguish between H_0 and H_3. Therefore, *privacy* of SCALES holds for Construction 2. □

7 Applications of RGS and iDRE

We outline certain other applications for the cryptographic objects we define.

7.1 RGS for Outsourced Re-Garbling

Consider a setting where a party P_{fun} holding a private function f would like to let a client P_{eval} securely evaluate $f(x)$ on various inputs x of its choice, using a GC-based protocol. Because of the one-time nature of GCs, this requires P_{fun} to carry out garbling once for each evaluation. This motivates the problem of *outsourced re-garbling* – i.e., out-sourcing the task of creating many copies of a garbled circuit for a private function to a semi-honest server (say, a cloud service).

Outsourced Re-Garbling presents an immediate application of RGS. The following definition of the Outsourced Re-Garbling task captures the security requirement that the parties P_{fun} and P_{eval} learn nothing more than in the original two-party setting, while a regarbling server S_{gb} that P_{fun} interacts with (before P_{eval} arrives) would learn nothing about the function f (except a permitted leakage $\phi(f)$). The security guarantees below assume that the server S_{gb} does not collude with P_{eval}.

Definition 14. *An **Outsourced Re-Garbling** scheme for a function family $\mathcal{F}$ with input domain $\mathcal{X}$ and a leakage function $\phi : \mathcal{F} \to \{0,1\}^*$, is a tuple of PPT algorithms* $(\mathsf{InitGb}, \mathsf{ReGb}, \mathsf{En}, \mathsf{Ev})$ *that satisfy the following properties:*

– ***Correctness:*** $\forall f \in \mathcal{F}$, $\forall x \in \mathcal{X}$,

$$\Pr[\mathsf{Ev}(F, X) = f(x) \ : (F_0, e) \leftarrow \mathsf{InitGb}(f), \\ (F, \pi) \leftarrow \mathsf{ReGb}(F_0), X \leftarrow \mathsf{En}(x, \pi(e))] = 1$$

– ***Privacy against*** S_{gb}*:* $\forall f \in \mathcal{F}$, *there exists a PPT simulator* $\mathsf{Sim}_{\mathsf{gb}}$ *such that*

$$\{\mathsf{Sim}_{\mathsf{gb}}(\phi(f))\} \overset{c}{\approx} \{F_0\}_{(F_0,e)\leftarrow\mathsf{InitGb}(f)}$$

– ***Privacy against*** P_{eval}*:* $\forall f \in \mathcal{F}$, $\forall n \in \mathbb{N}, \forall i \in [n]$ *and* $\forall x_i \in \mathcal{X}$, *there exists a PPT simulator* $\mathsf{Sim}_{\mathsf{eval}}$ *such that*

$$\{\mathsf{Sim}_{\mathsf{eval}}(\{f(x_i), x_i\}_{i\in[n]}, \phi(f))\} \overset{c}{\approx} \{\{F_i, X_i\}_{i\in[n]}\}_{\substack{(F_0,e)\leftarrow\mathsf{InitGb}(f), \\ \{(F_i,\pi_i)\leftarrow\mathsf{ReGb}(F_0), \\ X_i\leftarrow\mathsf{En}(x_i,\pi_i(e))\}_{i\in[n]}}}$$

These algorithms can be employed by the parties P_{fun}, P_{eval} and S_{gb} as follows. P_{fun} first executes $(F_0, e) \leftarrow \mathsf{InitGb}(f)$ and sends F_0 to S_{gb}. Then S_{gb} runs multiple instances of $(F_i, \pi_i) \leftarrow \mathsf{ReGb}(F_0)$ and sends all π_i back to P_{fun}. When P_{eval} comes online with an input x_i to f, it first gets F_i directly from S_{gb} (so P_{fun} does not incur the corresponding communication overhead). It then participates in a protocol with P_{fun} to obtain $X_i \leftarrow \mathsf{En}(x_i, \pi_i(e))$; this can be implemented directly using parallel OTs. Following that, P_{eval} computes $f(x_i) \leftarrow \mathsf{Ev}(F_i, X_i)$.

Note that the computational and communication complexity of P_{fun} involves a single instance of InitGb, followed by n instances of computing $\pi_i(e)$ and n instances of carrying out En. There is an implicit efficiency requirement that the latter two steps (which are repeated n times each) depend linearly on the *input size* m and are independent of its *circuit size* $|f|$, reducing the computational complexity of P_{fun} from $O(|f|n)$ to $O(|f| + mn)$ (ignoring factors involving the security parameter). This is a significant saving when $|f|$ and n are both large (e.g., evaluating a large machine learning model on inputs from the user-base of a popular app).

Theorem 4. *An RGS* $\mathsf{GS} = (\mathsf{Gb}, \mathsf{Rerand}, \mathsf{En}, \mathsf{Ev})$ *(Definition 8) is an Outsourced Re-Garbling scheme* $(\mathsf{InitGb}, \mathsf{ReGb}, \mathsf{En}, \mathsf{Ev})$ *(Definition 14).*

The proof directly follows from correctness and privacy properties of an RGS, except that for privacy against P_{eval} a standard hybrid argument is used to argue that multiple instances of regarbled circuits are simultaneously indistinguishable from multiple instances of freshly garbled circuits.

Proof. The fact that $(\mathsf{InitGb}, \mathsf{ReGb}, \mathsf{En}, \mathsf{Ev})$, when initialized with $\mathsf{GS} = (\mathsf{Gb}, \mathsf{Rerand}, \mathsf{En}, \mathsf{Ev})$, satisfies *Correctness* follows directly from the *Correctness* and Rerand-*Privacy* (Definition 8) of the RGS, as indicated in Section 4.

Claim 5. $\mathsf{GS} = (\mathsf{Gb}, \mathsf{Rerand}, \mathsf{En}, \mathsf{Ev})$ *satisfies* P_{fun}*-privacy (against* S_{gb}*).*

Proof. In this corruption case S_{gb} is corrupted and we are required to protect the privacy of P_{fun}'s input, f, from it. The view of the adversary is the message α that it receives and $\phi(f)$ that it may infer from α. It has no private inputs.

In order to show that P_{fun}-*privacy* is preserved, we define $\mathsf{Sim}_{\mathsf{gb}}(\phi(f))$ to pick an arbitrary $f' \in \mathcal{F}$ such that $\phi(f') = \phi(f)$ and then execute $(F', e') \leftarrow \mathsf{Gb}(f')$. We designate $F' = \alpha'$ as the output of $\mathsf{Sim}_{\mathsf{gb}}$. The fact that this view is indistinguishable from $\alpha \leftarrow \mathsf{InitGb}(f)$ can be claimed via a reduction to the *privacy* of garbling schemes (Definition 3).

Namely, consider for the sake of contradiction that there exists a PPT distinguisher D that, given $\phi(f)$, and a message α can tell weather α came from a real execution or was output by $\mathsf{Sim}_{\mathsf{gb}}(\phi(f))$ with non-negligible advantage ϵ. Then D can be used to construct a PPT distinguisher D' for breaking the GS privacy (Definition 3) as follows:

- D' samples functions $f_0 = f$ and $f_1 = f'$, and inputs x_0 and x_1 such that $f_0(x_0) = f_1(x_1)$ and $\phi(f_0) = \phi(f_1)$.
- D' sends $\phi(f_0)$ to D and sends f_0,f_1,x_0 and x_1 to the challenger C.
- C samples $b \in \{0, 1\}$, creates $(F_b, e_b) \leftarrow \mathsf{Gb}(f_b)$ and $X_b \leftarrow \mathsf{En}(x_b, e_b)$, and sends F_b, X_b to D'.
- D' sends F_b to D and outputs whatever D outputs.

Note that when $b = 0$ this is distributed as in the real execution since F_0 is a garbling of f and when $b = 1$, this is distributed as in the simulation created by $\mathsf{Sim}_{\mathsf{gb}}$. D' has the same advantage ϵ as D in breaking the privacy of the underlying garbling scheme which violates the security of GS. Consequently, no such distinguishers D' and D can exist, and P_{fun}-*privacy* is preserved. □

Claim 6. $\mathsf{GS} = (\mathsf{Gb}, \mathsf{Rerand}, \mathsf{En}, \mathsf{Ev})$ *satisfies* P_{fun}*-privacy (against* P_{eval}*).*

Proof. In this corruption case P_{eval} is corrupted and we are required to protect P_{fun}'s input f from it. The view of the adversary is its set of inputs $\{x_i\}_{i\in[n]}$, the messages $\{F_i\}_{i\in[n]}$ obtained from S_{gb} and the outputs $\{X_i\}_{i\in[n]}$ obtained from F_{En}. It then evaluates each of these to get $\{f(x_i)\}_{i\in[n]}$.

In order to show that P_{fun}-*privacy* is preserved, $\mathsf{Sim}_{\mathsf{eval}}(\{f(x_i), x_i\}_{i\in[n]}, \phi(f))$ is defined to do the following: for each $i \in [n]$, pick $f'_i \in \mathcal{F}$ such that $\phi(f'_i) = \phi(f)$ and $f'_i(x_i) = f(x_i)$. Run $(F'_i, e'_i) \leftarrow \mathsf{Gb}(f'_i)$, and $X'_i \leftarrow \mathsf{En}(x_i, e'_i)$; and output the set $\{(F'_i, X'_i)\}_{i\in[n]}$.

In order to show that this distribution is indistinguishable from the real execution, we define a sequence of $2n + 1$ hybrid games where each is comprised out of n pairs of (F_i, X_i), a garbling and an input encoding. The first hybrid H_0 is the simulated distribution output by $\mathsf{Sim}_{\mathsf{eval}}$.
In the next n hybrids, H_i is defined such that the first i pairs $(F_1, X_1) \cdots (F_i, X_i)$ are generated as fresh garblings corresponding to the same secret function f,

whereas the remaining pairs $(F_{i+1}, X_{i+1}) \cdots (F_n, X_n)$ are generated as fresh garblings for different functions f_j as in the simulation. The hybrid H_n consists of n pairs of fresh garblings $\{F_i, X_i\}_{i\in[n]}$ of f with different inputs x_i being encoded as X_i.
In the last n hybrids, H_{n+i} is defined such that the first i pairs $(F_1, X_1) \cdots (F_i, X_i)$ are generated as rerandomized garblings corresponding to f, whereas the remaining pairs $(F_{i+1}, X_{i+1}) \cdots (F_n, X_n)$ are generated as fresh garblings for f. Note that H_{2n} is distributed as in the real execution.

Consider for the sake of contradiction that there exists a PPT distinguisher D that, given $\phi(f)$ and $\{f(x_i), x_i\}_{i\in[n]}$, and a set $\{(F_i, X_i)\}_{i\in[n]}$ can say weather it came from a real execution or the $\{(F_i, X_i)\}_{i\in[n]}$ was output by $\mathsf{Sim}_{\mathsf{eval}}$ with non-negligible advantage ϵ. Then there must exist an index i for which D can distinguish H_i and H_{i-1} with an advantage at least $\frac{\epsilon}{2n}$.

If $i \leq n$, then such a D can be used as a subroutine by D', a PPT distinguisher for *Privacy* of GS (Definition 3) as follows:

- D' samples samples an index $i \in [n]$ uniformly at random. It then picks a function $f_0 = f$.
- Next, D' picks $f_1 \in \mathcal{F}$ such that $\phi(f_0) = \phi(f_1)$, and inputs x_0 and x_1 such that $f_0(x_0) = f_1(x_1)$.
- D' sends f_0,f_1,x_0 and x_1 to the challenger C.
- D' also $n - i$ other functions $f^{i+1}, \cdots, f^n \in \mathcal{F}$ such that for each such function f^j, $\phi(f) = \phi(f^j)$. It sets $f^1, \cdots, f^{i-1}$ equal to f. Finally, it samples $n-1$ other inputs $x^1, \cdots, x^{i-1}, x^{i+1}, \cdots, x^n \in \{0,1\}^m$ and sets $x^i = x_0$. The latter inputs $x^{i+1}, \cdots, x^n$ are picked such that each $f^j(x^j) = f(x^j)$.
- D' sends $(\{f(x^j), x^j\}_{j\in[n]}, \phi(f))$ to D.
- The challenger C samples $b \in \{0,1\}$. Then it creates $(F_b, e_b) \leftarrow \mathsf{Gb}(f_b)$ and $X_b \leftarrow \mathsf{En}(x_b, e_b)$. It sends (F_b, X_b) to D'.
- D' creates the set H by putting in the i^{th} position, the challenge (F_b, X_b). For every other position j, it generates $(F_j, e_j) \leftarrow \mathsf{Gb}(f^j)$ and $X_j \leftarrow \mathsf{En}(x^j, e_j)$. The final set $H = \{F_j, X_j\}_{j\in[n]}$ is given to D. Note that for $b = 0$ this is H_i since (F_0, X_0) are also formed from f_0. For $b = 1$ this is H_{i-1}.
- Finally, D' outputs whatever D outputs.

D' has advantage $\frac{\epsilon}{2n}$, since this is the advantage that D wold have in distinguishing the hybrids. However, since *privacy* for GS holds, there can exist no such D' there can exist no such D.

For $i > n$, D can be used as a subroutine by D'', a PPT distinguisher for Rerand-*Privacy* of GS (Definition 8) as follows:

- D' samples samples an index $i \in [n]$ uniformly at random. It then picks a function f, and a set of inputs $\{x_j\}_{j\in[n]}$ where each $x_j \in \{0,1\}^m$.
- D' sends $(\{f(x^j), x^j\}_{j\in[n]}, \phi(f))$ to D.
- Next, D' gives f, x_i to the challenger C.
- C first samples randomness r and uses r to create a prior garbling: $(F, e) \leftarrow \mathsf{Gb}(f, r)$. Next, it samples a bit $b \in \{0,1\}$. If $b = 0$, it creates $(F_0, \pi_{\mathsf{En}}) \leftarrow \mathsf{Rerand}(F)$ and $X_0 = \mathsf{En}(x_i, \pi_{\mathsf{En}}(e)$. If $b = 1$, it creates a fresh garbling $(F_1, e_1) \leftarrow \mathsf{Gb}(f)$ and $X_1 = \mathsf{En}(x_i, e_1)$. Then (r, F_b, X_b) are sent to D'.

– D' creates the set H by putting in the i^{th} position, the challenge (F_b, X_b). For every position $j < i$, it generates $(F'_j, e_j) \leftarrow \mathsf{Gb}(f)$, $(F_j, \pi_{\mathsf{En}}) \leftarrow \mathsf{Rerand}(F'_j)$ and $X_j \leftarrow \mathsf{En}(x_j, \pi_{\mathsf{En}}(e_j))$. For all $j > i$, it generates $(F_j, e_j) \leftarrow \mathsf{Gb}(f)$ and $X_j \leftarrow \mathsf{En}(x_j, e_j)$. The final set $H = \{F_j, X_j\}_{j\in[n]}$ is given to D. Note that for $b = 0$ this is H_{n+i} since (F_0, X_0) are rerandomized garblings. For $b = 1$ this is H_{n+i-1}.
– Finally, D' outputs whatever D outputs.

D' has advantage $\frac{\epsilon}{2n}$, since this is the advantage that D wold have in distinguishing the hybrids. However, since Rerand-*privacy* for GS holds, there can exist no such D' there can exist no such D. Therefore, P_{fun}-*privacy* is preserved. □

So, it follows that, $\mathsf{GS} = (\mathsf{Gb}, \mathsf{Rerand}, \mathsf{En}, \mathsf{Ev})$ satisfies Definition 14. □

7.2 iDRE for MPC

An iDRE can be used to implement a general n-party protocol under static semi-honest corruption of up to $n-1$ parties. Let $P_1, \cdots, P_n$ be the parties, f be the public function and $x \in \{0,1\}^m$ be its input out of which each P_i possesses $x^i \subset x$. The iDRE-based protocol can compute $f(x)$ using $O(n \times m)$ string-OT calls, meeting the lower bound on OT complexity for this setting, as proven in [HIK07]. This is achieved by letting each P_i act as one of the encoders in the sequential process along with playing the role of an input party. All the parties first employ PreEn and then every pair of parties engages in an OT for every input bit. Next, starting from P_1, the incremental chain of encoding follows with each P_i creating B_i and passing it on to P_{i+1}. Finally, P_n passes $\{s_i\}_{i\in[m]}$ to all other parties. Each party runs Combine for each of their input bits. These results are passed back to P_n that decodes and broadcasts the output.

Theorem 5. *Let* $(\mathsf{PreEn}, \mathsf{En}, \mathsf{En}^*, \mathsf{Combine}, \mathsf{Dec})$ *be an iDRE (Definition 13) for the function family* $\mathcal{F}$ *where* $f \in \mathcal{F}$ *has domain* $\{0,1\}^m$*. Figure 3 is an* n*-party semi-honest protocol computing* f *in the (string) OT-hybrid under* $(n-1)$*-corruption using* $((n-1) \times m)$ *OT calls and the iDRE in a black-box way.*

Proof. The *correctness* of the protocol follows from the correctness of the iDRE: there exists a decoder Dec such that $\mathsf{Dec}(\hat{f}_0(r), \{\hat{f}_i(x_i, r)\}_{i\in[m]}) = f(x)$.

We now show that the protocol in Figure 3 is secure in the OT-hybrid model. Let OT be the two party functionality through which OT is carried out in the OT-hybrid. For $m_0, m_1 \in \{0,1\}^*$ and $b \in \{0,1\}$, $(\perp, m_b) \leftarrow \mathsf{OT}((m_0, m_1), b)$. Let S be the sender with input (m_0, m_1), and let R be the receiver with input b that receives m_b from OT. Let Adv be the semi-honest PPT adversary that controls a subset of $(n-1)$ of the n parties and let $\mathcal{P}_{j^*}$ be the remaining honest party.

In the ideal model, let F be the trusted functionality computing f. It takes inputs $x^{i'}$ from each party $\mathcal{P}_{i'}$ and returns to $\mathcal{P}_n$ the value $f(x)$. Let Sim be the simulator in this model for the view of Adv. Sim has access to a simulator Sim′ that exists by definition for the iDRE of f. It knows the input of the semi-honest adversary: $\{x_i\}_{x_i \in x - x^n}$ which it gives as input to F.

n party protocol for t=n-1 corruption

$f : \{0,1\}^m \to \{0,1\}^l$ is a public function to be computed
$(\mathsf{PreEn}, \mathsf{En}, \mathsf{En}^*, \mathsf{Combine}, \mathsf{Dec})$ is the iDRE for f
for input $x \in \{0,1\}^m$, the subset of bits $x^{i'} \subset x, |x^{i'}| = m_{i'}$ is $\mathcal{P}_{i'}$'s private input

for $\mathcal{P}_j \in \{\mathcal{P}_1, \cdots, \mathcal{P}_n\}$ **do**
  $\mathcal{P}_j$ executes $\mathsf{PreEn}(r_j; j) \to \{e^0_{ij}, e^1_{ij}\}_{i\in[m]}$

for $\mathcal{P}_{i'} \in \{\mathcal{P}_1, \cdots, \mathcal{P}_n\}$ **do**
  for $x_i \in x^{i'}$ **do**
    for $\mathcal{P}_j \in \{\mathcal{P}_1, \cdots, \mathcal{P}_n\} - \{\mathcal{P}_{i'}\}$ **do** ▷ get shares
      $\mathcal{P}_{i'}$ performs OT with $\mathcal{P}_j$:
        Receiver $\mathcal{P}_{i'}$'s input choice bit is x_i
        Sender $\mathcal{P}_j$'s input strings are e^0_{ij}, e^1_{ij}
        $\mathcal{P}_{i'}$ receives $e^{x_i}_{ij}$

$\mathcal{P}_1$ executes $\mathsf{En}(r_1) \to B_1$
for $\mathcal{P}_j \in (\mathcal{P}_2, \cdots, \mathcal{P}_n)$ **do** ▷ actions for each party $\mathcal{P}_j$
  $\mathcal{P}_j$ gets B_{j-1} from $\mathcal{P}_{j-1}$
  $\mathcal{P}_j$ executes $\mathsf{En}^*(r_j; j, B_{j-1}) \to B_j$

for $\mathcal{P}_{i'} \in \{\mathcal{P}_1, \cdots, \mathcal{P}_n\}$ **do**
  for $x_i \in x^{i'}$ **do**
    get $s_i \in B_n$ from $\mathcal{P}_n$
    execute $\mathsf{Combine}(\{e^{x_i}_{ij}\}_{j\in[n]}, s_i) \to Z_i$
  send $Z'_{i'} = \{Z_i\}_{x_i \in x^{i'}}$ to $\mathcal{P}_n$

$\mathcal{P}_n$ gets $Z'_1, \cdots, Z'_{n-1}$ ▷ actions for $\mathcal{P}_n$
$\mathcal{P}_n$ has Z'_n, B_n
$\mathcal{P}_n$ executes $\mathsf{Dec}(\hat{f}_0(r), \{Z_i\}_{i\in[m]}) \to f(x)$

Fig. 3 Semi-honest MPC protocol based on iDRE

Sim needs to simulate the view of Adv in the OT-hybrid and therefore, besides sending and receiving messages, also emulates the OT calls for Adv. In the protocol, for the OT calls in which Adv acts as S, it gives to Sim the messages (m_0, m_1) and receives nothing. For the OT calls in which Adv plays the role of R, it gives its choice bit b to Sim and receives m_b.

In the protocol, the view of Adv is different for the case where the honest party $\mathcal{P}_{j^*} = \mathcal{P}_n$, the last party, and where it is not. Therefore, Sim works differently for these two cases as well.

Case 1: $\mathcal{P}_n$ is honest and Adv *controls all the other parties.* For this case, the view of Adv contains the following:

- $\{x_i\}_{x_i \in x - x^n}$ - the input bits of the corrupt parties.
- $\{r_j\}_{j\in[n-1]}$ - the randomness of the corrupt parties.
- $\{e^{x_i}_{in}\}_{x_i \in x - x^n}$ - strings received from OT with $\mathcal{P}_n$ where Adv acts as R.
- $\{s_i\}_{x_i \in x - x^n}$ - the final share for each input bit of Adv from $\mathcal{P}_n$.

In order to simulate this in the ideal world, Sim works as follows:

– Sim possesses input bits $\{x_i\}_{x_i \in x - x^n}$. It picks uniformly at random the elements in $\{r_j\}_{j \in [n-1]}$. Then, Sim invokes Adv with these values.
– Corresponding to each input bit of the honest party, Sim emulates $(n-1)$ calls to OT where Adv is S. For each call, Sim receives (e^0_{ij}, e^1_{ij}) for the i^{th} input bit. Adv does not get an output.
– Sim honestly runs $\{e^0_{in}, e^1_{in}\}_{i \in [m]} \leftarrow \mathsf{PreEn}(r_n; n)$. Then for each bit $x_i \in x - x^n$, it emulates OT by receiving x_i from Adv and returning $e^{x_i}_{in}$.
– Sim receives B_{n-1} from Adv and honestly executes $B_n \leftarrow \mathsf{En}^*(r_n; n, B_{n-1})$.
– It gives Adv, the set $\{s_i\}_{x_i \in x - x^n} \in B_n$.
– Finally, Sim receives $\{Z_i\}_{x_i \in x - x^n}$ from Adv and terminates.

Note that the view generated by Sim here is exactly identical to that in the OT-hybrid model.

Case 2: Adv *controls all parties except for a party* $\mathcal{P}_{j^*} \neq \mathcal{P}_n$. For this case, the view of Adv consists of the following:

– $\{x_i\}_{x_i \in x - x^{j^*}}$ - the input bits of the corrupt parties.
– $\{r_j\}_{j \neq j^*}$ - the randomness of the corrupt parties.
– $\{e^{x_i}_{ij^*}\}_{x_i \in x - x^{j^*}}$ - strings received from OT with $\mathcal{P}_{j^*}$ where Adv acts as R.
– B_{j^*} - the intermediate encoding that Adv receives from $\mathcal{P}_{j^*}$.
– $\{Z_i\}_{x_i \in x^{j^*}}$ - the reconstructed encodings that $\mathcal{P}_{j^*}$ gives Adv.
– $f(x)$ - the final output that Adv learns on decoding.

In order to simulate this in the ideal world, Sim works as follows:

– Sim has the input bits $\{x_i\}_{x_i \in x - x^{j^*}}$. After passing this to F, it receives $f(x)$.
– Sim invokes the iDRE simulator using these:

$$\{r_j\}_{j \neq j^*}, B_{j^*}, \{e^{x_i}_{ij^*}\}_{x_i \in x - x^{j^*}}, \{Z_i\}_{x_i \in x^{j^*}} \leftarrow \mathsf{Sim}'(f(x), \{x_i\}_{x_i \in x - x^{j^*}})$$

Then, Sim invokes Adv with $\{x_i\}_{x_i \in x - x^{j^*}}$ and $\{r_j\}_{j \neq j^*}$.
– Corresponding to each input bit of the honest party, Sim emulates $(n-1)$ calls to OT where Adv is S. For each call, Sim receives (e^0_{ij}, e^1_{ij}) for the i^{th} input bit. Adv does not get an output.
– Then for each bit $x_i \in x - x^{j^*}$, Sim emulates OT by receiving x_i from Adv and returning $e^{x_i}_{ij^*}$.
– Sim receives B_{j^*-1} from Adv during the encoding phase.
– It gives back B_{j^*} to Adv. Then, Adv proceeds to perform the rest of the encoding with this as the basis until B_n is created.
– For each $x_i \in x^{j^*}$, Adv gives s_i to Sim.
– Adv receives $\{Z_i\}_{x_i \in x^{j^*}}$ from Sim.
– Finally, Adv performs the decoding process for $\mathcal{P}_n$ and gets $f(x)$.
– Thus, the simulated view consists of:

$$\{x_i\}_{x_i \in x - x^{j^*}}, \{r_j\}_{j \neq j^*}, \{e^{x_i}_{ij^*}\}_{x_i \in x - x^{j^*}}, B_{j^*}, \{Z_i\}_{x_i \in x^{j^*}}, f(x)$$

The differences between the simulated and the real views is that in the simulated view, the sets $\{r_j\}_{j \neq j^*}, B_{j^*}, \{e_{ij^*}^{x_i}\}_{x_i \in x - x^{j^*}}, \{Z_i\}_{x_i \in x^{j^*}}$ are all derived from the execution of Sim', whereas in the OT-hybrid model, these were created according to the protocol.
Assume, by contradiction, that there exists a PPT distinguisher D' that can distinguish between the real and simulated views of the protocol. D' receives as input a set $(\{x_i\}_{x_i \in x - x^{j^*}}, \{r_j\}_{j \neq j^*}, \{e_{ij^*}^{x_i}\}_{x_i \in x - x^{j^*}}, B_{j^*}, \{Z_i\}_{x_i \in x^{j^*}}, f(x))$, performs operations in polynomial time, and returns a bit b. If $b = 1$, D' decides that the view received is the real view and it is a simulated view otherwise. D' has non-negligible advantage ϵ in the given game.

Then we can construct a PPT distinguisher D that can break *iDRE privacy* by using D' as a subroutine. D gives $(\{x_i\}_{x_i \in x - x^{j^*}}, f(x))$ to a challenger and gets back $(\{r_j\}_{j \neq j^*}, \{e_{ij^*}^{x_i}\}_{x_i \in x - x^{j^*}}, B_{j^*}, \{Z_i\}_{x_i \in x^{j^*}})$. It needs to perform operations that take polynomial-time overall and returns a bit b such that $b = 0$ if D decides that the view received has been generated by Sim', the iDRE simulator, and is generated from a real execution of the iDRE functions otherwise. D works by simply passing all of $(\{x_i\}_{x_i \in x - x^{j^*}}, \{r_j\}_{j \neq j^*}, \{e_{ij^*}^{x_i}\}_{x_i \in x - x^{j^*}}, B_{j^*}, \{Z_i\}_{x_i \in x^{j^*}}, f(x))$ to D' and returning the bit that D' returns. As D' has ϵ advantage, D has the same advantage in its game.

Since *iDRE privacy* holds and no such D exists that has non-negligible advantage and therefore no such D' can exist.

Therefore, we have that the protocol in Figure 3 securely computes f in the OT-hybrid model in the presence of any semi-honest PPT adversary Adv corrupting up to $n-1$ parties.[12] Furthermore, as is evident from the protocol itself, no more than $((n-1) \times m)$ calls to OT are made. □

OT complexity. For each input bit $x_i \in x$, the party $\mathcal{P}_{i'}$ possessing it participates in OT with all other parties $\mathcal{P}_j$ in order to receive $\{e_{ij}^{x_i}\}_{j \neq i' \in [n]}$. It possesses $e_{ii'}^{x_i}$ as the encoder that created it. This corresponds to $n-1$ OT calls for each input bit. No other step in the protocol uses OT. Hence, the protocol uses no more than $((n-1) \times m)$ OT calls in all. The communication complexity for this protocol is $O(n\kappa|f| + \kappa n^2)$. While there exist other MPC protocols with better communication complexity, our protocol meets the lower-bound in the number of required OT calls (see discussion in Section 1.1). Further, our protocol is black-box in its use of the iDRE.

Acknowledgments

We thank Shai Halevi for discussions including feedback regarding the gap in [GHV10]. Anasuya Acharya and Carmit Hazay are supported by the BIU

[12] This protocol can be extended to have all parties (not only $\mathcal{P}_n$) learn the output. In such a scenario, each party can either locally act as a decoder, or $\mathcal{P}_n$ can send $f(x)$ to all other parties. This eliminates the need for separate cases in the security proof since a computationally indistinguishable simulated view of the adversary now can be generated by the iDRE simulator Sim for all cases.

Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office, and by ISF grant No. 1316/18. Vladimir Kolesnikov was supported in part by NSF award #1909769, by a Facebook research award, a Cisco research award, and by Georgia Tech's IISP cybersecurity seed funding (CSF) award. Manoj Prabhakaran is supported by a Ramanujan Fellowship of the Department of Science and Technology, India. Carmit Hazay and Manoj Prabhakaran are also supported by the Algorand Centres of Excellence programme managed by Algorand Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of Algorand Foundation.

References

ABT18. Benny Applebaum, Zvika Brakerski, and Rotem Tsabary. Perfect secure computation in two rounds. In *TCC*, pages 152–174, 2018.

BBD+20. Zvika Brakerski, Pedro Branco, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Constant ciphertext-rate non-committing encryption from standard assumptions. In *TCC*, pages 58–87, 2020.

BCG+18. Nir Bitansky, Ran Canetti, Sanjam Garg, Justin Holmgren, Abhishek Jain, Huijia Lin, Rafael Pass, Sidharth Telang, and Vinod Vaikuntanathan. Indistinguishability obfuscation for RAM programs and succinct randomized encodings. *SIAM J. Comput.*, 47(3):1123–1210, 2018.

BGG+20. Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. Can a public blockchain keep a secret? In *TCC*, pages 260–290, 2020.

BGSZ21. James Bartusek, Sanjam Garg, Akshayaram Srinivasan, and Yinuo Zhang. Reusable two-round MPC from LPN. *IACR Cryptol. ePrint Arch.*, page 316, 2021.

BHHO08. Dan Boneh, Shai Halevi, Michael Hamburg, and Rafail Ostrovsky. Circular-secure encryption from decision diffie-hellman. In *CRYPTO*, pages 108–125, 2008.

BHR12. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *CCS*, pages 784–796, 2012.

BJKL21. Fabrice Benhamouda, Aayush Jain, Ilan Komargodski, and Huijia Lin. Multiparty reusable non-interactive secure computation from LWE. In *EUROCRYPT*, pages 724–753, 2021.

BL18. Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In *EUROCRYPT*, pages 500–532, 2018.

BMR90. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *STOC*, pages 503–513, 1990.

CFGN96. Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *STOC*, pages 639–648, 1996.

CGG+21. Arka Rai Choudhuri, Aarushi Goel, Matthew Green, Abhishek Jain, and Gabriel Kaptchuk. Fluid MPC: secure multiparty computation with dynamic participants. In *CRYPTO*, pages 94–123, 2021.

GGHR14. Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In *TCC*, pages 74–94, 2014.

GHK+21. Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. YOSO: you only speak once - secure MPC with stateless ephemeral roles. In *CRYPTO*, pages 64–93, 2021.

GHM+21. Craig Gentry, Shai Halevi, Bernardo Magri, Jesper Buus Nielsen, and Sophia Yakoubov. Random-index PIR and applications. In *TCC*, pages 32–61, 2021.

GHV10. Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. i-hop homomorphic encryption and rerandomizable yao circuits. In *CRYPTO*, pages 155–172, 2010.

GMPS21. Vipul Goyal, Elisaweta Masserova, Bryan Parno, and Yifan Song. Blockchains enable non-interactive MPC. *IACR Cryptol. ePrint Arch.*, page 1233, 2021.

GS18. Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In *EUROCRYPT*, pages 468–499, 2018.

HIK07. Danny Harnik, Yuval Ishai, and Eyal Kushilevitz. How many oblivious transfers are needed for secure multiparty computation? In *CRYPTO*, pages 284–302, 2007.

IK00. Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *FPCS*, pages 294–304, 2000.

LP09. Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *J. Cryptol.*, 22(2):161–188, 2009.

MRZ15. Payman Mohassel, Mike Rosulek, and Ye Zhang. Fast and secure three-party computation: The garbled circuit approach. In *SIGSAC*, pages 591–602, 2015.

NS09. Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO*, pages 18–35, 2009.

RS21. Rahul Rachuri and Peter Scholl. Le mans: Dynamic and fluid MPC for dishonest majority. *IACR Cryptol. ePrint Arch.*, page 1579, 2021.

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

YKT19. Yusuke Yoshida, Fuyuki Kitagawa, and Keisuke Tanaka. Non-committing encryption with quasi-optimal ciphertext-rate based on the DDH problem. In *ASIACRYPT*, pages 128–158, 2019.

A Instantiating KMHE for Garbling

Strong KMHE (Definition 11) can be instantiated by existing schemes like the one given in [BHHO08]. In our instantiation, we restrict the more general cryptosystem in [BHHO08] to a limited key and message space and functionality. Along with being a CPA secure encryption scheme, this scheme also is λ-*key leakage resilient* as shown in [NS09]. This property is defined as follows:

Definition 15. ***(key-leakage attacks)*** *A public-key encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is semantically secure against* $\lambda(n)$*-key-leakage attacks if for any probabilistic polynomial-time* $\lambda(n)$*-key-leakage adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *it holds that*

$$\mathsf{Adv}^{\mathsf{Leakage}}_{\Pi,\mathcal{A}}(n) = \left| \Pr[\mathsf{Expt}^{\mathsf{Leakage}}_{\Pi,\mathcal{A}}(0) = 1] - \Pr[\mathsf{Expt}^{\mathsf{Leakage}}_{\Pi,\mathcal{A}}(1) = 1] \right|$$

is negligible in n*, where for the challenger's secret* $b \leftarrow \{0, 1\}$*, the experiment* $\mathsf{Expt}^{\mathsf{Leakage}}_{\Pi,\mathcal{A}}(b)$ *is defined as follows:*

1. $(SK, PK) \leftarrow \mathsf{Gen}(1^n)$
2. $(M_0, M_1, \mathsf{state}) \leftarrow \mathcal{A}_1^{\mathsf{Leakage(SK)}}(PK)$ *such that* $|M_0| = |M_1|$
3. $C \leftarrow \mathsf{Enc}_{PK}(M_b)$
4. $b' \leftarrow \mathcal{A}_2(C, \mathsf{state})$
5. *output* b'*.*

Let $\mathbb{G}$ be a cyclic group of order q and $g \in \mathbb{G}$ be a generator of $\mathbb{G}$. For a negligible function δ, let the key length parameter $\kappa = \lambda + 2\log q + 2\log\frac{1}{\delta}$ where λ is a leakage parameter, and let $\mathbb{S}^\kappa$ be the set of all permutations on κ positions.

Informally, the secret key sk is restricted for the domain of κ-bit strings with $\frac{\kappa}{2}$ 0s and $\frac{\kappa}{2}$ 1s. Let $\alpha_0, \alpha_1 \in \mathbb{G}$ be two arbitrarily picked elements that represent the numeric values for 0 and 1 respectively. Then, the message space consists of plaintexts $m = (m_1, \cdots, m_\kappa)$ where each $m_i \in \{\alpha_0, \alpha_1\}$ and m contains exactly $\frac{\kappa}{2}$ α_0's and $\frac{\kappa}{2}$ α_1's. During encryption, loosely speaking, first a set of public keys $\mathsf{pk} = (\mathsf{pk}_1, \cdots, \mathsf{pk}_\kappa)$ are sampled where $\mathsf{pk}_i \in \mathbb{G}^{\kappa+1}$ is derived from sk. Then pk_i is used to create $c_i \in \mathbb{G}^{\kappa+1}$ for each $m_i \in m$. The final ciphertext is $c = (c_1, \cdots, c_\kappa)$.

The message space transformations $g \in \mathcal{F}_{msg}$ is the set of permutations over κ length strings. A transformation is applied to a ciphertext $c = (c_1, \cdots, c_\kappa)$ by permuting the elements in c according to g. Note that in the ciphertext, each $c_i \in c$ was computed for a bit $m_i \in m$. Therefore permuting the c_i elements results in permuting the bits $m_i \in \mathcal{M}$. Let M_g be the permutation matrix for g. Such a permutation is therefore applied by computing $M_g \times c$.

The key space transformations $f \in \mathcal{F}_{key}$ is also the set of permutations over κ length strings. In order to apply this, the first κ elements in each vector $c_i \in c$ are permuted according to f, effectively rearranging the order in which the public key elements were used to compute the ciphertext. Therefore, letting M_f be the permutation matrix for f, such a permutation is therefore applied by computing $c \times M_f$ for the first κ columns of c.

The Eval algorithm is used to apply both the key space and message space permutations to a ciphertext. Along with applying the permutation matrices to the ciphertext rows and columns, it also performs an additional step of *blinding* the ciphertext. That is, given a ciphertext c and a product $c' = \mathsf{Eval}(c, f, g)$, c' would look like a freshly created ciphertext. This is carried out by sampling a random element $r_i \in \mathbb{Z}_q$ for each row c_i of the ciphertext. Next, for each $\mathsf{pk}_i \in \mathsf{pk}$, each element in pk_i is raised to the power of r_i. Finally, elements in the resulting vector are multiplied to elements in $c_i \in c$ that have in the same positions.

Formally, sharable KMHE is instantiated with [BHHO08] as follows:

Construction 3. *Let $\mathbb{G}$ be a cyclic group and let $\alpha_0, \alpha_1 \in \mathbb{G}$. Let $\mathbb{S}^\kappa$ be the set of all permutations on κ positions. The sharable KMHE scheme* $\mathsf{KMH} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval}, \mathsf{Share}, \mathsf{Recon})$ *instantiated using [BHHO08] is defined as follows:*

- *$\mathcal{K}$ is the set of all $(s_1, \cdots, s_\kappa) \in \{0,1\}^\kappa$ with $\frac{\kappa}{2}$ 0s and the rest 1s*
- *$\mathcal{M}$ is the set $\mathbb{G}^\kappa$. Among these, the set $\mathcal{M}_\mathcal{K} = \{(m_1, \cdots, m_\kappa) \in \{\alpha_0, \alpha_1\}^\kappa\}$ with $\frac{\kappa}{2}$ α_0's and the rest α_1's represent $\mathcal{K}$.*
- *$\mathcal{C} = \mathbb{G}^{(\kappa+1)\times\kappa}$ is a set of matrices of elements in $\mathbb{G}$*
- *$\mathcal{F}_{key} = \mathbb{S}^\kappa$ and for $f \in \mathcal{F}_{key}$, M_f is a binary sparse matrix that represents it.*
- *$\mathcal{F}_{msg} = \mathbb{S}^\kappa \cup \mathbb{G}^\kappa$ is the set of transformations that include $\mathbb{S}^\kappa$ where M_g is the binary matrix that represents it. $\mathcal{F}_{msg}$ also includes vectors in $\mathbb{G}^\kappa$ that can be added or subtracted from a message $m \in \mathbb{G}^\kappa$. For such transformations let $h \in \mathbb{G}^\kappa$ be a vector representing it.*

These are used to instantiate the following functions:

- $\mathsf{sk} \leftarrow \mathsf{Gen}(1^\kappa)$*:*
  - *sample* $\mathsf{sk} \leftarrow \mathcal{K}$
  - *output -* $\mathsf{sk} = (s_1, \cdots, s_\kappa)$
- $\mathsf{pk} \leftarrow \mathsf{PKgen}(\mathsf{sk})$*:*
  - *input -* $\mathsf{sk} = (s_1, \cdots, s_\kappa) \leftarrow \mathcal{K}$
  - *sample generators $g_1, \cdots, g_\kappa \leftarrow \mathbb{G}$ and compute $h = \Pi_{i=1}^{\kappa} g_i^{s_i}$*
  - *output -* $\mathsf{pk} = (g_1, \cdots, g_\kappa, h)$
- $c \leftarrow \mathsf{Enc}(\mathsf{sk}, m)$*:*
  - *input - key* $\mathsf{sk} = (s_1, \cdots, s_\kappa) \leftarrow \mathcal{K}$
  - *input - message $m = (m_1, \cdots, m_\kappa) \in \mathcal{M}$*
  - *$\forall m_i \in m$, sample $r_i \leftarrow \mathbb{Z}_q$, compute $\mathsf{pk}_i = (g_{1,i}, \cdots, g_{\kappa,i}, h) \leftarrow \mathsf{PKgen}(\mathsf{sk})$ and $c_i = (g_{1,i}^{r_i}, \cdots, g_{\kappa,i}^{r_i}, h^{r_i} \cdot m_i)$*
  - *output - $c = (c', \mathsf{pk})$ where $c' = (c_1, \cdots, c_\kappa)$ and $\mathsf{pk} = (\mathsf{pk}_1, \cdots, \mathsf{pk}_\kappa)$*
- $m = \mathsf{Dec}(\mathsf{sk}, c)$*:*
  - *input - key* $\mathsf{sk} = (s_1, \cdots, s_\kappa) \leftarrow \mathcal{K}$
  - *input - ciphertext $c = (c', \mathsf{pk})$*
  - *parse $c' = (c_1, \cdots, c_\kappa) \in \mathcal{C}$*
  - *for each $c_i = (u_1, \cdots, u_\kappa, e) \in c'$, compute $m_i = e \cdot \left(\Pi_{i=1}^{\kappa} u_i^{s_i}\right)^{-1}$*
  - *output - $m = (m_1, \cdots, m_\kappa)$*
- $c' \leftarrow \mathsf{Eval}(c, f, g)$*:*

- *input - permutation matrix* M_f *for* $f \in \mathcal{F}_{key}$
- *input - permutation matrix* M_g *for* $g \in \mathbb{S}^\kappa \subset \mathcal{F}_{msg}$ *or offset vector* $h \in \mathbb{G}^\kappa \subset \mathcal{F}_{msg}$

- *input - ciphertext* $c' = (c, \mathsf{pk})$
- *let* $c = (c_1, \cdots, c_\kappa) \in \mathcal{C}$
- *let* $\mathsf{pk} = (\mathsf{pk}_1, \cdots, \mathsf{pk}_\kappa)$
- *if* M_g *is the input, compute* $c'' = M_g \times c$. *otherwise, for* $h = (h_1, \ldots, h_\kappa)$ *compute* c'' *by multiplying each* h_i *to the last element in* $c_i \in c$ *for all* $i \in [\kappa]$
- *compute* $c'' = c'' \times M_f$ *to the first* κ *columns, for the key transformation*
- *sample* $r = (r_1, \cdots, r_\kappa) \leftarrow \mathbb{Z}_q^\kappa$
- *for each* $\mathsf{pk}_i = (\alpha_1, \cdots, g_\kappa, h) \in \mathsf{pk}$*, for each* $c_i = (\alpha_{1,i}, \cdots, \alpha_{\kappa,i}, \alpha_{\kappa+1,i}) \in c''$*, compute* $c'_i = (\alpha_{1,i} \cdot \alpha_1^{r_i}, \cdots, \alpha_{\kappa,i} \cdot g_\kappa^{r_i}, \alpha_{\kappa+1,i} \cdot h^{r_i})$
- *let* $c = (c'_1, \cdots, c'_\kappa) \in \mathcal{C}$
- *let* $\mathsf{pk} = (r_1 \cdot \mathsf{pk}_1, \cdots, r_\kappa \cdot \mathsf{pk}_\kappa)$
- *output -* $c' = (c, \mathsf{pk})$

– $([m]_0, [m]_1) \leftarrow \mathsf{Share}(m)$*:*
  - *input - message* $m \in \{\alpha_0, \alpha_1\}^\kappa$ *containing* $\frac{\kappa}{2}$ α_0*'s and the rest* α_1*'s*
  - *sample* $[m]_0 \leftarrow \mathbb{G}^\kappa$ *and compute* $[m]_1 \in \mathbb{G}^\kappa$ *such that* $\forall i \in [\kappa]$*, for* $m_i \in m, m_i^0 \in [m]_0, m_i^1 \in [m]_1$*,* $m_i = m_i^0 \cdot m_i^1$
  - *output -* $([m]_0, [m]_1)$

– $m = \mathsf{Recon}([m]_0, [m]_1)$*:*
  - *input - messages* $[m]_0, [m]_1 \in \mathbb{G}^\kappa$
  - $\forall i \in [\kappa]$*, for* $m_i^0 \in [m]_0, m_i^1 \in [m]_1$*, compute* $m_i = m_i^0 \cdot m_i^1$
  - *output -* $m = (m_1, \ldots, m_\kappa)$

Informally, KMH privacy, as in Definition 11, requires indistinguishability between two ciphertexts of known messages encrypted under two different keys, for one of which, the adversary knows some additional information. It is already proven in [NS09] that assuming the hardness of DDH, the [BHHO08] encryption scheme is resilient against key leakage attacks as in Definition 15. We use this to prove the following theorem:

Theorem 6. *Assuming the hardness of DDH in* $\mathbb{G}$*, the [BHHO08] encryption scheme as in Construction 3 is a sharable KMHE scheme that satisfies Definition 12.*

Proof. Construction 3 satisfies *Correctness* and *KMH Correctness* by definition.

Claim 7. *Construction 3 satisfies Key Privacy.*

Proof. The key space $\mathcal{K}$ contains all κ-bit strings with $\frac{\kappa}{2}$ 0's and $\frac{\kappa}{2}$ 1's. Let $k, k' \in \mathcal{K}$ be such keys. The key domain transformation family $\mathcal{F}_{key}$ contains all permutations over κ positions. Let $f \leftarrow \mathcal{F}_{key}$ be picked uniformly at random. In order for *Key Privacy* to hold, we require that,

$$\{k, k'\}_{k \leftarrow \mathsf{Gen}(1^\kappa), k' \leftarrow \mathsf{Gen}(1^\kappa)} \overset{s}{\approx} \{k, f(k)\}_{k \leftarrow \mathsf{Gen}(1^\kappa), f \leftarrow \mathcal{F}_{key}}$$

In Construction 3,

$$\Pr[(k,k') : k = k_0 \leftarrow \mathsf{Gen}(1^\kappa), k' = k_1 \leftarrow \mathsf{Gen}(1^\kappa)] = \frac{1}{|\mathcal{K}| \cdot |\mathcal{K}|}$$

since $\mathsf{Gen}(1^\kappa)$ samples a key uniformly at random from the key space. Furthermore, a permutation on the positions within the key can map a given key to any other key with equal probability. Therefore,

$$\Pr[(k,k') : k = k_0 \leftarrow \mathsf{Gen}(1^\kappa), f \leftarrow \mathcal{F}_{key}, k' = f(k)] = \frac{1}{|\mathcal{K}| \cdot |\mathcal{K}|}$$

Thus for a key pair (k,k'), both these distributions are identical and *Key Privacy* holds. □

Claim 8. *Construction 3 satisfies KMH Privacy.*

Proof. We assume, by contradiction, that there exists a PPT adversary Adv that can distinguish in the *KMH privacy* game with non-negligible advantage ϵ. Adv gets the keys $k_0, k_1, f(k_1) \in \mathcal{K}$ from the challenger. Letting the total number of queries be t, Adv samples $\{m_i, m'_i \in \mathcal{M}, g_i \in \mathcal{F}_{msg}\}_{i\in[t]}$. For each such instance (m_i, m'_i, g_i), it forwards to the challenger the triple (c_i, g_i, m'_i) where $c = \mathsf{Enc}(k_0, m_i)$ using the encryption algorithm of Construction 3. Adv receives a ciphertext c^i_b each time, and finally it needs to output its guess b' for b.

The proof for *KMH privacy* follows via a sequence of hybrids for the view of the adversary. These hybrids are as follows:

- $H_0 = \{k_0, k_1, f(k_1), \{m_i, m'_i, g_i, c_i, \mathsf{Enc}(f(k_0), g_i(m_i))\}_{i\in[t]}\}$: this view corresponds to the case in the KMH privacy game when $b = 0$. The function $f \in \mathcal{F}_{key}$ is a secret chosen by the challenger. The keys $k_0, k_1, f(k_1)$ are given by the challenger in the beginning of the game. Each tuple (m_i, m'_i, g_i) are sampled by the adversary and c_i is also created by it using the encryption algorithm from Construction 3. Each ciphertext $\mathsf{Enc}(f(k_0), g_i(m_i))$ is rerandomized by the challenger from c_i.
- $H_1 = \{k_0, k_1, f(k_1), \{m_i, m'_i, g_i, c_i, \mathsf{Enc}(f(k_0), m^i_1)\}_{i\in[t]}\}$: in this hybrid, the elements $(k_0, k_1, f(k_1), \{m_i, m'_i, g_i, c_i\}_{i\in[t]})$ are sampled and computed in the same way as in H_0. However, each ciphertext $\mathsf{Enc}(f(k_0), m^i_1)$ is created differently: for the i^{th} query, the challenger samples a fresh m^i_1 uniformly at random and this is kept secret from Adv. Then m^i_1 is encrypted using $f(k^0)$ using the encryption algorithm from Construction 3.
- $H_2 = \{k_0, k_1, f(k_1), \{m_i, m'_i, g_i, c_i, \mathsf{Enc}'(f(k_0), m^i_1)\}_{i\in[t]}\}$: in this hybrid, again, the elements $(k_0, k_1, f(k_1), \{m_i, m'_i, g_i, c_i\}_{i\in[t]})$ are sampled and computed in the same way as in H_0 and H_1. But each challenge ciphertext $\mathsf{Enc}'(f(k_0), m^i_1)$ created by first sampling a fresh m^i_1 uniformly at random that is kept secret from Adv. Then encryption is performed using an algorithm $\mathsf{Enc}'(\cdot,\cdot)$. This is a modified encryption function where for a message $m \in \mathcal{M}$ as in Construction 3, for each $m_i \in m$, κ different elements

$r_1, \cdots, r_\kappa \in_{u.a.r.} \mathbb{Z}_q$ are first sampled. Then, $c_i = (g_1^{r_1}, \cdots, g_\kappa^{r_\kappa}, h' \cdot m_i)$ where $h' = \Pi_{j=1}^{\kappa} g_j^{r_j \cdot s_j}$.

- $H_3 = \{k_0, k_1, f(k_1), \{m_i, m_i', g_i, c_i, \mathsf{Enc}'(k', m_1^i)\}_{i\in[t]}\}$: this hybrid is identical to the previous hybrid except, again, for the way in which each challenge ciphertext $\mathsf{Enc}'(k', m_1^i)$ is created. Here, the challenger samples a fresh key $k' \leftarrow \mathsf{Gen}(1^\kappa)$ that is used to encrypt *all* the challenges (instead of using $f(k_0)$). This key is independent of $(k_0, k_1, f(k_1))$ that Adv is given. For the i^{th} query, a fresh message m_1^i is sampled uniformly at random and kept secret from Adv. It is encrypted under k' using $\mathsf{Enc}'(\cdot,\cdot)$ as in the previous hybrid.
- $H_4 = \{k_0, k_1, f(k_1), \{m_i, m_i', g_i, c_i, \mathsf{Enc}(k', m_1^i)\}_{i\in[t]}\}$: this hybrid is identical to H_3 except that each challenge ciphertext $\mathsf{Enc}(k', m_1^i)$ is created using the encryption algorithm as in Construction 3.
- $H_5 = \{k_0, k_1, f(k_1), \{m_i, m_i', g_i, c_i, \mathsf{Enc}(k', m_i')\}_{i\in[t]}\}$: this view corresponds to the case in the KMH privacy game when $b = 1$. From the previous hybrid, we switch here, for each challenge ciphertext, from an encryption of an unknown random m_1^i, to that of m_i' chosen by Adv.

If Adv is a PPT adversary that can distinguish between hybrids H_0 and H_5 with a non-negligible advantage ϵ, it then follows that one of the adjacent hybrids can be distinguished with advantage at least $\frac{\epsilon}{5}$. The proof for the theorem follows from the fact that each pair of adjacent hybrids can be shown as indistinguishable:

1. The fact that $H_0 \stackrel{c}{\approx} H_1$ can be reduced to security against λ-key leakage attacks (Definition 15). Intuitively, we require that given *leakage* $k_0, k_1, f(k_1)$ for some secret key $f(k_0)$ of the challenger, any PPT adversary Adv' cannot distinguish between two ciphertexts computed as $c_0^i = \mathsf{Enc}(f(k_0), g_i(m_i))$ and $c_1^i = \mathsf{Enc}(f(k_0), m_1^i)$ for an unknown m_1^i that is sampled uniformly at random by the challenger.
2. Proving that $H_1 \stackrel{c}{\approx} H_2$ and $H_3 \stackrel{c}{\approx} H_4$ are both shown in a similar manner by a reduction to the DDH problem: the difference between these adjacent hybrids stems from the fact that in one, a Diffie-Hellman tuple is used in the encryption algorithm for each challenge ciphertext and this is not the case for the other. Furthermore, since the ciphertexts encrypt a message m_1^i that was sampled uniformly at random and is unknown to Adv, it follows that there could be multiple possible correct decryptions.
3. We argue that H_2 and H_3 are identically distributed. This is because,going to the latter hybrid from H_2 the encryption key is switched from $f(k_0)$ to a freshly sampled k'. For each unknown m_1^i chosen at random since, a ciphertext c_i created using $\mathsf{Enc}'(\cdot,\cdot)$ can potentially be decrypted to the same message for the public key $(g_1, \cdots, g_\kappa, h)$ and different values of $(r_1, \cdots, r_\kappa)$ in the exponent given that they satisfy the constraints on h. These constraints are the different secret keys of the form $(s_1, \cdots, s_\kappa)$.
4. Lastly, the fact that $H_4 \stackrel{c}{\approx} H_5$ can be reduced to the standard CPA security requirement satisfied by Construction 3.

First, in order to show that $H_0 \overset{c}{\approx} H_1$, we define a sequence of hybrids $J_0, \cdots, J_t$, each of the form,

$$J_i = \{k_0, k_1, f(k_1), \{m_j, m'_j, g_j, c_j\}_{j\in[t]}, \{\mathsf{Enc}(f(k_0), m_1^i)\}_{j\leq i}, \{\mathsf{Enc}(f(k_0), g_i(m_i))\}_{j>i}\}$$

Note that $J_0 = H_0$ and $J_t = H_1$. If the PPT distinguisher Adv that can distinguish for *KMH privacy* of Construction 3 with non-negligible advantage ϵ, can distinguish between H_0 and H_1, it follows that there must exist an index $i \in [t]$ for which it can distinguish between neighbouring hybrids J_{i-1} and J_i with a non-negligible advantage $> \frac{\epsilon}{5t}$. Such an Adv can be used as a subroutine to define a new PPT key-leakage adversary A_{leak}, that works as follows:

– A_{leak} first samples the keys $k_0, k_1 \leftarrow \mathsf{Gen}(1^\kappa)$.
– Letting C_{leak} be the key leakage challenger, C_{leak} samples a secret key $sk' \leftarrow \mathsf{Gen}(1^\kappa)$. It gives to A_{leak}, a corresponding public key $pk' \leftarrow \mathsf{PKgen}(sk')$.
– Next, A_{leak} defines a leakage function $\mathsf{leakage}_{k_0,k_1}(\cdot)$ that works as follows: on input $sk' \in \mathcal{K}$, first sample a function $f \leftarrow \mathcal{F}_{key}$ uniformly at random under the constraint that $sk' = f(k_0)$. Then output $f(k_1) \in \mathcal{K}$.
– A_{leak} sends $\mathsf{leakage}_{k_0,k_1}(\cdot)$ as a query to C_{leak} and gets back $f(k_1)$ as a response.
– A_{leak} then sends $k_0, k_1, f(k_1)$ to Adv. It also samples an index $i \in [t]$ uniformly at random.
– A_{leak} receives from Adv, the tuple $\{c_j = \mathsf{Enc}(k_0, m_j), g_j, m'_j\}_{j\in[t]}$, where each $m_j, m'_j \in \mathcal{M}$ and $g_j \in \mathcal{F}_{msg}$ is sampled by Adv.
– Now, A_{leak} creates the challenge ciphertext set as follows:
  - It samples a fresh message $m'_1 = m_1^i \leftarrow \mathcal{M}$ uniformly at random. It sets $m'_0 = g_i(\mathsf{Dec}(k_0, c_i)) = g_i(m_i)$. Then, (m'_0, m'_1) are sent to the challenger C_{leak}.
  - It gets back a challenge ciphertext c_b. Note that when $b = 0$, $c_0 = \mathsf{Enc}(f(k_0), g_i(m_i))$ that is equivalent to $\mathsf{Eval}(c, f, g)$ by definition in Construction 3. Otherwise, $c_1 = \mathsf{Enc}(f(k_0), m_1^i)$.
  - It appends the public key to the ciphertext, making $c'_b = (c_b, pk')$. This is set as c^i.
  - For all $j < i$, A_{leak} samples a fresh $m_1^j \in \mathcal{M}$ uniformly at random and encrypts m_1^j with pk, getting $c^j = \mathsf{Enc}(f(k_0), m_1^j)$.
  - For all $j > i$, A_{leak} first decrypts c_j using k_0 to get $g_j(m_j) \in \mathcal{M}$. It then encrypts the message with pk, getting $c^j = \mathsf{Enc}(f(k_0), g_j(m_j))$.
– A_{leak} sends the set $\{c^j\}_{j\in[t]}$ to Adv.
– Finally, A_{leak} outputs whatever Adv outputs.

It follows that conditioned on the correctness of Eval and the fact that $\mathsf{leakage}_{K_0,K_1}(\cdot)$ is a valid leakage function according to [NS09], A_{leak} has advantage $> \frac{\epsilon}{5t}$ in this game, which is non-negligible. However, since no such A_{leak} can exist for [BHHO08], no such Adv can distinguish for any index between J_i and J_{i-1}. Therefore, $H_0 \overset{c}{\approx} H_1$.

Note that Eval is correct by definition in Construction 3. The leakage function $\mathsf{leakage}_{k_0,k_1}(\cdot)$ is valid since we set the permissible leakage to $\lambda = \kappa - 2\log q -$

$2\log\frac{1}{\epsilon} = \kappa(1-o(1))$ in the construction. The fact that $\mathsf{leakage}_{k_0,k_1}(\cdot)$ reveals no more than λ bits of the secret key sk' to A_{leak} is derived from the lemma in [GHV10]:

Lemma 3. *Let $L_1, L_2 \in HW_{\kappa,\kappa/2}$ and $\pi \in S_\kappa$ be chosen uniformly at random. Then,*

$$\tilde{H}_\infty(\pi(L_1)|L_1, L_2, \pi(L_2)) \geq \kappa - \frac{3}{2}\log\kappa$$

Here, $HW_{\kappa,k}$ denote the set of all κ bit strings with hamming weight k, and S_κ denotes the set of all permutations over κ elements. $\mathsf{leakage}_{k_0,k_1}(\cdot)$ gives us $\lambda' = \frac{3}{2}\log\kappa$ bits of information. This is less than λ for large enough κ. Therefore, $\mathsf{leakage}_{k_0,k_1}(\cdot)$ is a valid leakage function for Construction 3.

In order to show for the set of hybrids, $H_1 \overset{c}{\approx} H_2$, and $H_3 \overset{c}{\approx} H_4$, we would require a similar sequence of hybrids as in the above argument. We show now that $H_1 \overset{c}{\approx} H_2$, and a similar argument can be made to show $H_3 \overset{c}{\approx} H_4$. Consider a sequence of hybrids $J_0, \cdots, J_t$, each of the form,

$$J_i = \{k_0, k_1, f(k_1), \{m_j, m'_j, g_j, c_j\}_{j\in[t]}, \{\mathsf{Enc}'(f(k_0), m_1^i)\}_{j\leq i}, \{\mathsf{Enc}(f(k_0), m_1^i)\}_{j>i}\}$$

Note that $J_0 = H_1$ and $J_t = H_2$. If the PPT distinguisher Adv that can distinguish for *KMH privacy* of Construction 3 with non-negligible advantage ϵ, can distinguish between H_1 and H_2, it follows that there must exist an index $i \in [t]$ for which it can distinguish between neighbouring hybrids J_{i-1} and J_i with a non-negligible advantage $> \frac{\epsilon}{5t}$. Such an Adv can be used as a subroutine to define a distinguisher A_{DDH} for the Decisional Diffie Hellman problem, that works as follows:

- A DDH challenger gives A_{DDH} a tuple (a, b, c, d) and the goal is to output a bit b' such that if $b' = 1$, the challenge is a DH tuple (of the form g_1, g_2, g_1^r, g_2^r) and it is a non-DH tuples (of the form $g_1, g_2, g_1^{r_1}, g_2^{r_2}$) otherwise.
- A_{DDH} works by first sampling $k_0, k_1 \leftarrow \mathsf{Gen}(1^\kappa)$ and $f \leftarrow \mathcal{F}_{key}$ uniformly at random. It sends $k_0, k_1, f(k_1)$ to Adv.
- A_{DDH} gets from Adv the tuple $\{c_j = \mathsf{Enc}(k_0, m_j), g_j, m'_j\}_{j\in[t]}$, where each $m_j, m'_j \in \mathcal{M}$ and $g_j \in \mathcal{F}_{msg}$ is sampled by Adv.
- A_{DDH} samples a position $i \in [t]$ uniformly at random and creates the set of responses to Adv as follows:
  - For each j^{th} query, it samples a fresh message $m_1^j \in \mathcal{M}$ uniformly at random.
  - Then, for all $j < i$, it samples κ different elements $r_1^j, \cdots, r_\kappa^j \in_{u.a.r.} \mathbb{Z}_q$. Let $f(k_0) = (s_1, \cdots, s_\kappa)$. Then, for each element $m_k \in m_1^j$, it computes $c_k = (g_1^{r_1^j}, \cdots, g_\kappa^{r_\kappa^j}, h' \cdot m_k)$ where $h' = \Pi_{j=1}^{\kappa} g_j^{r_j^k \cdot s_j}$. The resulting ciphertext $c^j = (c_1, \cdots, c_\kappa)$ is an encryption of m_1^j under the key $f(k_0)$ using the algorithm $\mathsf{Enc}'(\cdot,\cdot)$ that uses a non-DDH tuple.
  - For all $i < j$, it creates $c^j = \mathsf{Enc}(f(k_0), m_1^i)$.

- Lastly, for the i^{th} response, it first extends the challenge from the DDH challenger to $(g_1, \cdots, g_\kappa, g_1^{r_1}, \cdots, g_\kappa^{r_\kappa})$. In the public key, h and h' are computed using these along with $f(k_0)$ and this is used to encrypt m_1^i. Note that if this is a DH tuple, then the c^i formed is according to the algorithm $\mathsf{Enc}(\cdot,\cdot)$ is as in Construction 3. Otherwise, it is as in $\mathsf{Enc}'(\cdot,\cdot)$.

– A_{DDH} sends the set $\{c^j\}_{j\in[t]}$ to Adv.
– Finally, A_{DDH} outputs whatever Adv outputs.

A_{DDH} has advantage $> \frac{\epsilon}{5t}$ in this game, which is non-negligible. However, since no such A_{DDH} can exist for [BHHO08], no such Adv can distinguish for any index between J_i and J_{i-1}. Therefore, $H_2 \overset{c}{\approx} H_1$.

We argue next that H_2 and H_3 are distributed identically when switching between secret keys $f(K_0)$ and K'. For each query $i \in [t]$, the challenger samples a plaintext $m_1^i \leftarrow \mathcal{M}$ uniformly at random. Therefore, for a challenge ciphertext c^i that is formed using $\mathsf{Enc}'(\cdot,\cdot)$, there could exist multiple valid decryptions to different plaintexts using different secret keys, each case being equally probable. This is possible since a non-DDH tuple allows different values $(r_1, \cdots, r_\kappa)$ in the exponent for the public key elements. This makes for a more general constraint for calculating h in $pk = (g_1, \cdots, g_\kappa, h)$ that can be satisfied by different secret keys for different values of $(r_1, \cdots, r_\kappa)$ in the exponent. In fact, fixing any message m_1, and corresponding h' (due to c being fixed), for each $K = (s_1, \cdots, s_\kappa)$, there is an $(r_1, \cdots, r_\kappa)$ that satisfies $h' = \Pi_{i=1}^{\kappa} g_i^{r_i \cdot s_i}$ in $\mathsf{Enc}'(\cdot,\cdot)$ for a fixed $(g_1, \cdots, g_\kappa, h)$. Therefore, $\Pr[f(K_0)|c] = \Pr[K'|c]$.

Finally, we claim that if Adv can distinguish between H_4 and H_5 with non-negligible advantage $\frac{\epsilon}{5}$, it can be used by a PPT adversary A_{CPA} to break the CPA security of Construction 3. Consider a sequence of hybrids $J_0, \cdots, J_t$, each of the form,

$$J_i = \{k_0, k_1, f(k_1), \{m_j, m'_j, g_j, c_j\}_{j\in[t]}, \{\mathsf{Enc}(k', m'_i)\}_{j\leq i}, \{\mathsf{Enc}(k', m_1^i)\}_{j>i}\}$$

Note that $J_0 = H_4$ and $J_t = H_5$. If the PPT distinguisher Adv that can distinguish for *KMH privacy* of Construction 3 with non-negligible advantage ϵ, can distinguish between H_4 and H_5, it follows that there must exist an index $i \in [t]$ for which it can distinguish between neighbouring hybrids J_{i-1} and J_i with a non-negligible advantage $> \frac{\epsilon}{5t}$. Such an Adv can be used as a subroutine to define a distinguisher A_{CPA}, that works as follows:

– A_{CPA} first samples the keys $k_0, k_1, k' \leftarrow \mathsf{Gen}(1^\kappa)$ and the transformation $f \leftarrow \mathcal{F}_{key}$. It gives $k_0, k_1, f(k_1)$ to Adv.
– Letting C_{CPA} be the challenger, C_{CPA} samples a secret key $sk' \leftarrow \mathsf{Gen}(1^\kappa)$. It gives to A_{CPA}, a corresponding public key $pk' \leftarrow \mathsf{PKgen}(sk')$.
– A_{CPA} then samples an index $i \in [t]$ uniformly at random.
– It receives from Adv, the tuple $\{c_j = \mathsf{Enc}(k_0, m_j), g_j, m'_j\}_{j\in[t]}$, where each $m_j, m'_j \in \mathcal{M}$ and $g_j \in \mathcal{F}_{msg}$ is sampled by Adv.
– Now, A_{CPA} creates the challenge ciphertext set as follows:

  - It samples a fresh message $m'_1 = m^i_1 \leftarrow \mathcal{M}$ uniformly at random. It sets $m'_0 = m'_i$. Then, (m'_0, m'_1) are sent to the challenger.
  - It gets back a challenge ciphertext c_b. Note that when $b = 0$, $c_0 = \mathsf{Enc}(k', m'_i)$ and, otherwise, it is $c_1 = \mathsf{Enc}(k', m^i_1)$.
  - It appends the public key to the ciphertext, making $c'_b = (c_b, pk')$. This is set as c^i.
  - For all $j > i$, A_{CPA} samples a fresh $m^j_1 \in \mathcal{M}$ uniformly at random and encrypts m^j_1 with pk, getting $c^j = \mathsf{Enc}(k', m^j_1)$.
  - For all $j < i$, A_{CPA} encrypts the message m'_j with pk, getting $c^j = \mathsf{Enc}(k', m'_j)$.
- A_{CPA} sends the set $\{c^j\}_{j\in[t]}$ to Adv.
- Finally, A_{CPA} outputs whatever Adv outputs.

Assuming the hardness of DDH, Construction 3 is CPA secure and therefore no such Adv can exist. □

Implementing Sharable KMHE. In order to construct Sharable KMHE, we need the following additional properties:

1. **Secret-Sharing Plaintext Labels.** In strong KMHE, for designated elements $\alpha_0, \alpha_1 \in \mathbb{G}$ each plaintext $m \in \mathcal{M} = \mathcal{K}$ is a message $m = (m_1, \cdots, m_\kappa)$ where each $m_i \in \{\alpha_0, \alpha_1\}$ and m contains exactly $\frac{\kappa}{2}$ α_0's and the rest α_1's. For sharable KMHE, we extend $\mathcal{M}$ to be $\mathbb{G}^\kappa$. For a message $m \in \mathcal{K}$, its sharing is computed as $\mathsf{Share}(m)$:
$$[m]_0 \leftarrow \mathbb{G}^\kappa$$
$$[m]_1 \in \mathbb{G}^\kappa \text{ s.t. } m = [m]_0 \circ [m]_1$$
where $\circ$ is the Hadamard product of the two vectors. In order to reconstruct m, one can compute $\mathsf{Recon}([m]_0, [m]_1) = [m]_0 \circ [m]_1 \in \mathbb{G}^\kappa$. These functions are defined in Construction 3.
2. **Refreshing Plaintext Shares.** For strong KMHE it suffices to restrict $\mathcal{F}_{key} = \mathcal{F}_{msg}$ to be the set of permutations over κ length strings. However the complete scheme in [BHHO08] allows for the application of any affine function over the message vector. Now, let $\mathcal{F}^*_{msg}$ be the space of function indexed by vectors $h \in \mathbb{G}^\kappa$,
$$h([m]_0) = [m]_0 \circ h$$
$$\overline{h}([m]_1) = [m]_1 \circ \overline{h}$$
such that $h \circ \overline{h} = I^\kappa$, where $I \in \mathbb{G}$ is the identity element. It is immediate that reconstructing using these shares gives us m. Note that for a vector h, its inverse in $\mathbb{G}^\kappa$ can be easily computed and so both functions in $\mathcal{F}^*_{msg}$ are defined when one is sampled. This function is also an affine function over the elements in any message $m \in \mathcal{M}$ and so can be homomorphically applied on the message space in [BHHO08]. The resulting message space function family for sharable KMHE is taken to be, $\mathcal{F}_{msg} = \mathcal{F}_{key} \cup \mathcal{F}^*_{msg}$.

For this it holds that, $\forall \sigma \in \mathcal{F}_{key}$, $\forall h \in \mathcal{F}^*_{msg}$, $\exists \overline{h} \in \mathcal{F}^*_{msg}$ s.t. $\forall k \in \mathcal{K}$,

$$\begin{aligned} &\big\{[k]_0, [k]_1, [\sigma(k)]_0, [\sigma(k)]_1\big\}_{\substack{([k]_0,[k]_1)\leftarrow \mathsf{Share}(k);\\ ([\sigma(k)]_0,[\sigma(k)]_1)\leftarrow \mathsf{Share}(\sigma(k))}} \\ \equiv &\big\{[k]_0, [k]_1, h(\sigma([k]_0)), \overline{h}(\sigma([k]_1))\big\}_{([k]_0,[k]_1)\leftarrow \mathsf{Share}(k)} \end{aligned}$$

Therefore, assuming DDH, [BHHO08] as in Construction 3 is a sharable KMHE scheme as in Definition 12. □

B A Gap in the Analysis of [GHV10]

The notion of a rerandomizable garbled circuit (RGC) was introduced in [GHV10], where they define *rerandomizable SFE* (Definition 7) and implement it using an RGC (though a formal definition for RGC was not given).[13] The RGC was then instantiated using the encryption scheme of [BHHO08]. Although their construction of a rerandomizable SFE is secure, it turns out that their proof has a slight gap, which we identify here.

First, we briefly recall the structure of the RGC in [GHV10]. Each garbled gate of the RGC carries, for each pair of values for the gate's input wires, encryptions of additive shares of output labels using the two input labels as keys. This construction, similar to Yao's garbled circuit construction, admits the simulation of the GC and the set of labels for a single input, based only on the circuit's output on that input (Theorem 7 in [GHV10]). This relies only on the semantic security of the encryption scheme used.

Next, a rerandomization procedure is provided: To rerandomize the GC, the bits of the labels are permuted, using independent permutations for each wire, but the same permutation for both values of a wire. Towards this, the authors exploit two properties of the encryption scheme of [BHHO08]. Firstly, it allows such transformations to be applied to both the key and the message of a ciphertext, and secondly, as established by Naor and Segev [NS09], it is resistant to leakage from the secret key. Gentry et al. insightfully note that the leakage-resilience property aligns with the goal of securing a rerandomizable garbled circuit, when the encryption keys used in the rerandomized garbled circuit could be correlated with keys of the original circuit which are known to the adversary. This requires that the keys become sufficiently random when a key-transformation is applied, and [GHV10] ensure this using a clever choice of the key-space and the transformation-space.

However, their proof does not match their security definition of a Rerandomizable SFE, which requires that a rerandomized second message in a 2-round SFE protocol is indistinguishable from a freshly generated one. In particular, it is shown that even given the original GC, the rerandomized GC is simulable, thanks to semantic security of the BHHO scheme under leakage-resilience, as defined by [NS09]. However, this semantic security applies only to indistinguishability of

[13] Rerandomizable SFE was defined as an intermediate notion in [GHV10], whose main contribution was a "multi-hop homomorphic encryption" primitive.

two ciphertexts which are both encrypted under the same (transformed) key. As such, it does not rule out the ability of an adversary to identify if a ciphertext was encrypted using a key obtained by transforming a known key, or from a fresh key. As we shall see, such ability would render their rerandomization scheme insecure; nevertheless, the encryption scheme of [BHHO08] enjoys additional properties that rule out this attack.

Below, in appendix B.1, we shall see an example of an encryption scheme which enjoys all the properties that are explicitly used in the proof in [GHV10], yet fails to meet their definition of rerandomizable SFE. We also present a definition of Key-and-Message Homomorphic Encryption, which explicitly models a key-privacy requirement, in addition to message-privacy. Further, following the proof in [NS09], we prove that the construction in [BHHO08] satisfies this definition. This lets us complete the proof of security of the rerandomizable SFE construction in [GHV10].

B.1 A Counter-Example

Note that Definition 15 of key-leakage resilience from [NS09] guarantees indistinguishability between encryptions of two messages under the *same* key, given that λ bits of the key are leaked.

Now, we provide an example of an encryption scheme which satisfies the above definition, as well as supports key-and-message homomorphism as relied upon by [GHV10]. We start from an encryption scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ that is semantically secure and λ-key leakage resilient (this can be thought of as the scheme in [BHHO08]). Let $\mathcal{K}$ be the key domain and $\mathcal{F}_{key}$ be the key transformation space for $\mathcal{E}$.

We use this to create another scheme $\mathcal{E}' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ which simply has an extra bit in the secret-key that is included in the ciphertext during encryption. The transformation space for the keys is expanded as described below.[14]

- $\mathsf{Gen}'(1^n) \to k \in \mathcal{K}'$ gives an n-bit key $k = (a, y)$ where $a \in \{0, 1\}$, and $y \leftarrow \mathsf{Gen}(1^{n-1})$ is $n - 1$ bits long.
- $\mathsf{Enc}'(k, m) = (\mathsf{Enc}(y, m), a)$ encrypts m using y and appends a to the result.
- $\mathsf{Dec}'(k, c) = \mathsf{Dec}'((a, y), (c', a)) = \mathsf{Dec}(y, c')$ decrypts c' using y, ignoring a.
- $\mathcal{E}'$ is homomorphic in the key domain for a function family $\mathcal{F}'_{key}$ with functions $F_{d,f} \in \mathcal{F}'_{key}$ where $f \in \mathcal{F}_{key}$ and $d \in \{0, 1\}$, such that $F_{d,f}(k) = F_{d,f}(a, y) = (a \oplus d, f(y))$.
- The message space contains the key space, and the scheme is homomorphic in the message domain for the same function family $\mathcal{F}'_{key}$ as above. For this, we assume that $\mathcal{E}$ has such a message space and message transformation space (which is the case for the scheme in [BHHO08]).

[14] [GHV10] relies on the specifics of their transformation space only to ensure that (1) it is supported by the encryption scheme, and (2) the leakage on a key $f(k_1)$, where f is drawn from the transformation space, by $(k_0, k_1, f(k_0))$ is bounded, so that the semantic security guarantee by [NS09] applies. As we shall see, these properties remain intact.

The following claim is easy to immediately verify:

Claim 9. *If $\mathcal{E}$ is λ-key leakage resilient, then $\mathcal{E}'$ is also λ-key leakage resilient.*

Further, [GHV10] lower bound the average conditional min-entropy $\widetilde{H}_\infty(f(k_1) \mid k_0, k_1, f(k_0))$, where $k_0, k_1 \leftarrow \mathcal{K}$ and $f \leftarrow \mathcal{F}_{key}$, so that the key leakage resilience of the encryption scheme $\mathcal{E}$ can be invoked (Lemma 9 in [GHV10]). We note that for keys $k'_0, k'_1 \leftarrow \mathcal{K}'$ and $F \leftarrow \mathcal{F}'_{key}$, $\widetilde{H}_\infty(F(k'_1) \mid k'_0, k'_1, F(k'_0)) = \widetilde{H}_\infty(f(k_1) \mid k_0, k_1, f(k_0))$ as above, and hence the same argument continues to apply for $\mathcal{E}'$.

Now we point out that a GC $\hat{\mathcal{C}}$, with input wire labels $\mathcal{L}$, that is created using $\mathcal{E}'$ as in the construction in [GHV10], and then rerandomized to $\hat{\mathcal{C}}_0$, does not meet the rerandomizable SFE definition. Specifically, $\hat{\mathcal{C}}_0$ and the keys corresponding to the labels for an input can be distinguished from a fresh GC, as these keys can be linked to the keys in the original GC.

Let $\mathrm{GHV}_{\mathcal{E}'}$ denote the GC construction of [GHV10], but instantiated using the encryption scheme $\mathcal{E}'$ (instead of the one in [BHHO08]). Below we refer to the garbling and rerandomization operations, Gb and Rerand in this scheme.

Claim 10. *Let $(\hat{\mathcal{C}}, \mathcal{L}) \in \mathrm{GHV}_{\mathcal{E}'}.\mathsf{Gb}(f)$. Then there exists a PPT distinguisher that, given $(\hat{\mathcal{C}}, \mathcal{L})$, can distinguish between $\hat{\mathcal{C}}_0 \leftarrow \mathrm{GHV}_{\mathcal{E}'}.\mathsf{Rerand}(\hat{\mathcal{C}})$ and $\hat{\mathcal{C}}_1 \leftarrow \mathrm{GHV}_{\mathcal{E}'}.\mathsf{Gb}(f)$, a fresh garbling of f with a positive constant advantage.*

Proof. Fix $\hat{\mathcal{C}}$. First consider a wire with keys $k_0, k_1 \in \mathcal{K}'$ where both keys have the first bit (the "extra" bit) equal. In $\hat{\mathcal{C}}_0$, a gate to which this wire is an input, includes four ciphertexts encrypted with keys $F(k_0)$ or $F(k_1)$, where $F \in \mathcal{F}'_{key}$ (encrypting shares of a key for the gate's output wire – but we shall be concerned only about the keys). Note that $F(k_0)$ and $F(k_1)$ will also have their first bits equal. Thus, all four of these ciphertexts will have the same first bit in $\hat{\mathcal{C}}_0$. However, in $\hat{\mathcal{C}}_1$, freshly sampled keys $k'_0, k'_1 \leftarrow \mathcal{K}'$ are used to create these ciphertexts, and their first bits are equal only with probability $1/2$. (Similarly, when k_0, k_1 from $\hat{\mathcal{C}}$ have different first bits, in $\hat{\mathcal{C}}_0$, these four ciphertexts will not all have the same first bit, but in $\hat{\mathcal{C}}_1$, this will happen with probability $1/2$.) Thus a distinguisher (which "knows" $\hat{\mathcal{C}}$) can distinguish between $\hat{\mathcal{C}}_0$ and $\hat{\mathcal{C}}_1$ with constant advantage. □

Therefore, even though the scheme $\mathcal{E}'$ satisfies both semantic security and λ-key leakage resilience, the rerandomizable SFE construction of [GHV10] is rendered insecure when its GC scheme is instantiated with $\mathcal{E}'$ instead of the encryption scheme of [BHHO08].

In our instantiation, we achieve rerandomizing using strong KMHE. The *KMH privacy* requirement explicitly requires that ciphertexts under an unknown freshly sampled key are indistinguishable from ciphertexts computed under a key that is rerandomized from a known prior key. This also holds given the prior keys k_0, k_1 and the new value of the (active) key k'_1.

Strong KMHE can be instantiated under the DDH hardness assumption based on the public-key encryption scheme from [BHHO08], and relying on its analysis from [NS09]. (Note that for simplicity, we have defined strong KMHE as a symmetric-key primitive; however, the definition can be extended to a public-key primitive naturally, and the scheme of [BHHO08] would satisfy this definition.) This is because not only is [BHHO08] semantically secure and secure against λ-key leakage resilience attacks, both of whose difficulty can be reduced to DDH, but it also holds that encryptions under any two keys are statistically indistinguishable from each other, even given certain leakage about one of the keys. Therefore, strong KMHE is *sufficient* for rerandomizing garbled circuits.

B.2 The Source of the Gap

What the proof in [GHV10] establishes is that given a garbled circuit $\hat{\mathcal{C}} \in \text{GHV}.\mathsf{Gb}(f)$, the rerandomized garbled circuit $\hat{\mathcal{C}}_0$ along with input labels for an input x, can be simulated using $f(x)$ (and not f itself). In this simulated garbled circuit, the *messages* in the ciphertexts that are part of the garbled gates are replaced; however, this simulated garbled circuit continues to use ciphertexts whose *keys* are as in $\hat{\mathcal{C}}_0$. These keys are correlated with the keys in $\hat{\mathcal{C}}$, and further the ciphertexts may reveal this correlation (as is the case in our counter-example). Thus $\hat{\mathcal{C}}_0$ (as well as the simulated garbled circuit) can be linked to $\hat{\mathcal{C}}$, contradicting the security requirement of Rerandomizable SFE that it should appear as a fresh garbled circuit.

Incidentally, as the proof in [GHV10] focused on the simulability of $\hat{\mathcal{C}}_0$, it does not refer to another basic requirement for a Rerandomizable SFE: since the evaluator learns one "active" key for each wire of the garbled circuit, this key should look like a fresh key independent of $\hat{\mathcal{C}}$. This is indeed satisfied by the choice of the key transformation space (of bit permutations of a balanced string) used in [GHV10], as well as in the counter-example we presented above.