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).

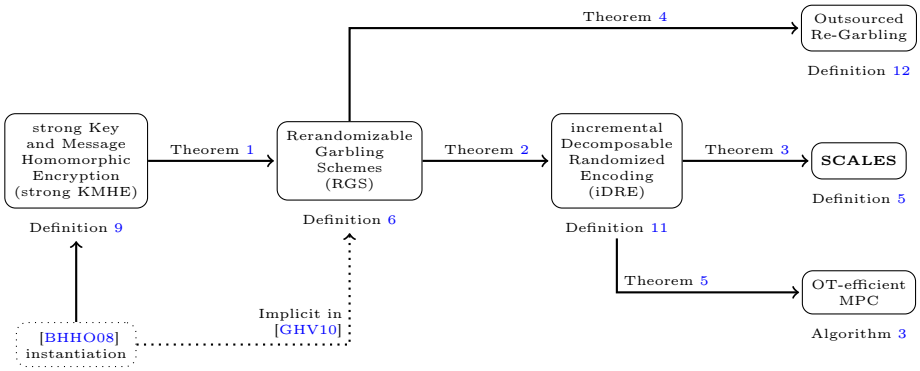

Fig. 1 Our contributions

• *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 1.4.

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 1.4 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).
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

[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.

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 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 5	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.

1.5 Technical Overview

We define and realize a new notion of randomized encodings [IK00] (Definition 3), 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 $\hat{C}'$ 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 $\hat{C}'$ are effectively secret shared between them. To evaluate $\hat{C}'$, 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

[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].

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 (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 the full version [AHKP22] 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.).

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 1. *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* $f_0(x_0) = f_1(x_1)$*,*

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

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 2. *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 1) 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 1 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 3. *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 4. *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 3) 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 2) is a DRE. Looking ahead, we extend Definition 4 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. A more detailed discussion can be found in the full version [AHKP22].

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.

Definition 5. *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):*

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

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 Aggregate are required to be independent of k (but may depend on the security parameter κ)[7]. This requirement on the complexity of InpEnc and 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 $\mathcal{B}_{j-1}$ and posts a message α_j on the board. After enough number of such iterations, each input player run Aggregate (Step 3) and post a message y_i. Finally, the function output is publicly derived using Decode (Step 4). The *privacy* guarantee

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

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 6. *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 1) 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 6 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 7. *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 2) for a family* $\mathcal{F}$ *of functions with input domain* $\{0,1\}^m$*, and* Rerand *is a PPT algorithm as in Definition 6 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 $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$,[8] 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 8. *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**:* $\forall m \in \mathcal{M}, k \in \mathcal{K}$*,*

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

- ***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* $\mathcal{A}$*):*

[8] 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.

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 9. *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 8 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 the full version [AHKP22]. 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 9 also satisfies Definition 8, 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 (but not the protocol!) of [GHV10] in the full version [AHKP22].

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 implicit in 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 9. We consider a special case of KMH with an additional structural property:

Definition 10. *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 9 for 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* $\mathcal{F}_{msg}$ *with the additional property that* $\mathcal{K} = \mathcal{M}$ *and* $\mathcal{F}_{key} = \mathcal{F}_{msg}$*.*

The scheme has two additional functions (1) $([k]_0, [k]_1) \leftarrow \mathsf{Share}(k)$ *that outputs two random shares of a key* $k \in \mathcal{K}$*. (2)* $k \leftarrow \mathsf{Recon}([k]_0, [k]_1)$ *that reconstructs the label* k *from its shares. These functions are such that the following property holds* $\forall \sigma \in \mathcal{F}_{key}$ *and* $\forall k \in \mathcal{K}$*,*

$$\mathsf{Share}(\sigma(k)) \equiv \{(\sigma([k]_0), \sigma([k]_1)\}_{([k]_0,[k]_1) \leftarrow \mathsf{Share}(k)}$$

We denote by $\mathsf{GS} = (\mathsf{Gb}_{\mathsf{KMH}}, \mathsf{En}_{\mathsf{KMH}}, \mathsf{Ev}_{\mathsf{KMH}}, \mathsf{Rerand}_{\mathsf{KMH}})$ a rerandomized garbling scheme where all the garbling scheme algorithms are instantiated with a sharable KMHE scheme KMH as the underlying encryption scheme. We next provide an overview of this garbling scheme:

The garbling algorithm $\mathsf{Gb}_{\mathsf{KMH}}(C, 1^\kappa)$ works as follows:

- For every wire $w_i \in \mathcal{W} - O$, sample labels $L^0_{w_i}, L^1_{w_i} \leftarrow \mathsf{Gen}(1^\kappa)$.
- 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 $g_i = (w_\ell, w_r, w_i, op) \in \mathcal{G}$, let $([L^b_{w_i}]_0, [L^b_{w_i}]_1)$ be the shares 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:

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

  these four rows are then permuted according to π.
- Output $\hat{\mathcal{C}} = ((G_1, \cdots, G_q), (\mathsf{L}_0, \mathsf{L}_1))$ and $\mathcal{L} = \{L^0_{w_i}, L^1_{w_i}\}_{w_i \in I}$.

An encoding algorithm $\mathsf{En}_{\mathsf{KMH}}(\mathcal{L}, x)$ gets a set of input labels $\mathcal{L}$ and the function input $x = (x_1, \cdots, x_m)$ and outputs $\mathcal{I} = \{L^{x_i}_{w_i}\}_{w_i \in I}$.

The evaluation algorithm $\mathsf{Ev}_{\mathsf{KMH}}(\hat{\mathcal{C}}, \mathcal{I})$ works gate by gate, by decrypting each row in the garbled gate.[9] The resulting plaintexts are combined to the output label using Recon. Evaluating a gate lets us derive one label for a wire in the

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

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 $f(x)$.

The rerandomizing algorithm $(\hat{\mathcal{C}}', \Pi) \leftarrow \mathsf{Rerand}_{\mathsf{KMH}}(\hat{\mathcal{C}})$ works as follows:

- For all wires $w_i \in \mathcal{W} - O$, sample $\sigma_i \in \mathcal{F}_{key}$.
- For all output wires $w_i \in O$, let σ_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. In order to rerandomize G_i into G'_i, the following is carried out:

$$G'_i = \begin{cases} (\mathsf{Eval}(c_{0,0}, \sigma_\ell, \sigma_i), \mathsf{Eval}(c_{0,1}, \sigma_r, \sigma_i)) \\ (\mathsf{Eval}(c_{1,0}, \sigma_\ell, \sigma_i), \mathsf{Eval}(c_{1,1}, \sigma_r, \sigma_i)) \\ (\mathsf{Eval}(c_{2,0}, \sigma_\ell, \sigma_i), \mathsf{Eval}(c_{2,1}, \sigma_r, \sigma_i)) \\ (\mathsf{Eval}(c_{3,0}, \sigma_\ell, \sigma_i), \mathsf{Eval}(c_{3,1}, \sigma_r, \sigma_i)) \end{cases} \quad \text{where } G_i = \begin{cases} (c_{0,0}, c_{0,1}) \\ (c_{1,0}, c_{1,1}) \\ (c_{2,0}, c_{2,1}) \\ (c_{3,0}, c_{3,1}) \end{cases}$$

  the rows in G'_i are permuted using π_i.
- 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 10). Then* $\mathsf{GS} = (\mathsf{Gb}_{\mathsf{KMH}}, \mathsf{Rerand}_{\mathsf{KMH}}, \mathsf{En}_{\mathsf{KMH}}, \mathsf{Ev}_{\mathsf{KMH}})$ *is an RGS with projective encoding (Definition 7).*

The detailed proof for this can be found in the full version [AHKP22].

5 Incremental Decomposable Randomized Encodings

In this section, we introduce a variant of Decomposable Randomized Encodings (DRE - Definition 4): an *incremental Decomposable Randomized Encoding* (iDRE). We also present a construction for an iDRE scheme based on an RGS, and a KMHE scheme (Definition 8). 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 11. *An **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 $x \in \{0,1\}^m$:*

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

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 $\mathcal{A} \subseteq [m]$,*

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

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 6) and KMHE scheme (Definition 8) 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.[10] This set of transformations is generated by the different instances of algorithm PreEn. At last, the $\mathsf{Combine}$ 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, $\mathcal{L}^j = \{L^b_{ij}\}_{i\in[m],b\in\{0,1\}}$, where $L^b_{ij} \in \{0,1\}^\kappa$ 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 $X^j = \{L^{x_i}_{ij}\}_{i\in[m]}$ for the input $x = \{x_i\}_{i\in[m]} \in \{0,1\}^m$.

The encrypted labels set that corresponds to F_j is denoted by $\mathcal{EL}^j$ where $\mathcal{EL}^j = \{\mathsf{Enc}(K^b_{ij}, L^b_{ij})\}_{i\in[m],b\in\{0,1\}}$. Starting with F_1, each label $L^b_{i1} \in \mathcal{L}^1$ is encrypted using a key K^b_{i1} that is chosen from a KMHE scheme. We represent by $\Pi\mathcal{K}^1 = \{K^b_{i1}\}_{i\in[m],b\in\{0,1\}}$ the set of these keys. Each subsequent $\mathcal{EL}^j$ is created from $\mathcal{EL}^{j-1}$. Namely, let $\rho^b_{ij} \in \mathcal{F}_{key}$ denote a transformation chosen to randomize the key ρ^b_{ij-1}, yielding a new transformed key ρ^b_{ij} in the key domain. Then $\Pi\mathcal{K}^j = \{\rho^b_{ij}\}_{i\in[m],b\in\{0,1\}}$ 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 8) and let* $\mathsf{GS} = (\mathsf{Gb}, \mathsf{Rerand}, \mathsf{En}, \mathsf{Ev})$ *be a projective RGS (Definition 7), then Figure 2 is an iDRE (Definition 11).*

The detailed proof for this theorem can be found in the full version [AHKP22].

[10] 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.

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**
    $e^b_{i1} = K^b_{i1} \leftarrow \mathsf{KMH.Gen}(1^\kappa; r^b_{ij})$ $\quad \forall i \in [m], b \in \{0,1\}$
  else
    $e^b_{ij} = \rho^b_{ij} \leftarrow \mathcal{F}_{key}$ using r^b_{ij} $\quad \forall i \in [m], b \in \{0,1\}$
  return $\{e^0_{ij}, e^1_{ij}\}_{i \in [m]}$

function $\mathsf{En}(f; r_1)$
  parse $r_1 = (r'_1, r^0_{11}, \cdots, r^0_{m1}, r^1_{11}, \cdots, r^1_{m1})$
  $\{K^b_{i1}\}_{i \in [m], b \in \{0,1\}} \leftarrow \mathsf{PreEn}(1; r_1)$
  $(F_1, e_1) \leftarrow \mathsf{GS.Gb}(f; r'_1)$
  $(L^b_{11}, \cdots, L^b_{m1}) \leftarrow \mathsf{GS.En}(b^m, e_1)$ $\quad \forall b \in \{0,1\}$
  $\alpha^b_{i1} \leftarrow \mathsf{KMH.Enc}(K^b_{i1}, L^b_{i1})$ $\quad \forall i \in [m], b \in \{0,1\}$
  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 $r_j = (r'_j, r^0_{1j}, \cdots, r^0_{mj}, r^1_{1j}, \cdots, r^1_{mj})$
  $\{\rho^b_{ij}\}_{i \in [m], b \in \{0,1\}} \leftarrow \mathsf{PreEn}(j; r_j)$
  $(F_j, \{\sigma_i\}_{i \in [m]}) \leftarrow \mathsf{GS.Rerand}(F_{j-1}; r'_j)$
  $\alpha^b_{ij} \leftarrow \mathsf{KMH.Eval}(\alpha^b_{ij-1}, \rho^b_{ij}, \sigma_i)$ $\quad \forall i \in [m], b \in \{0,1\}$
  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

6 Realizing SCALES

In Construction 1, 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 $e_{ij}^{x_i}$; 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 1. *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 11) 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* $\{e_{ij}^0, e_{ij}^1\}_{i \in [m]} = \mathsf{iDRE.PreEn}(j; r_j)$
  - *compute* $\forall i \in [m], m_2^{i,j} \leftarrow \mathsf{OT}_2(z_i, (e_{ij}^0, e_{ij}^1))$
  - *output* $\alpha_j = \{B_j, \{m_2^{i,j}\}_{i \in [m]}\}$
- $\forall i \in [m], y_i \leftarrow \mathsf{Aggregate}(\mathcal{B}_d, w_i)$ -
  - *compute* $\forall j \in [d], e_{ij}^{x_i} \leftarrow \mathsf{OT}_{out}(w_i, m_2^{i,j})$ *using* $m_2^{i,j} \in \mathcal{B}_d$
  - *output* $y_i = \mathsf{iDRE.Combine}(\{e_{ij}^{x_i}\}_{j \in [n]}, s_i)$ *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 11) 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 1 is a secure SCALES scheme (Definition 5).*

The detailed proof for this can be found in the full version [AHKP22].

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 12. *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*

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

- ***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*

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

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 6) is an Outsourced Re-Garbling scheme* $(\mathsf{InitGb}, \mathsf{ReGb}, \mathsf{En}, \mathsf{Ev})$ *(Definition 12).*

The detailed proof for this can be found in the full version [AHKP22].

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 $\mathsf{Combine}$ 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 11) 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.*

The proof for this theorem can be found in the full version [AHKP22]. 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 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

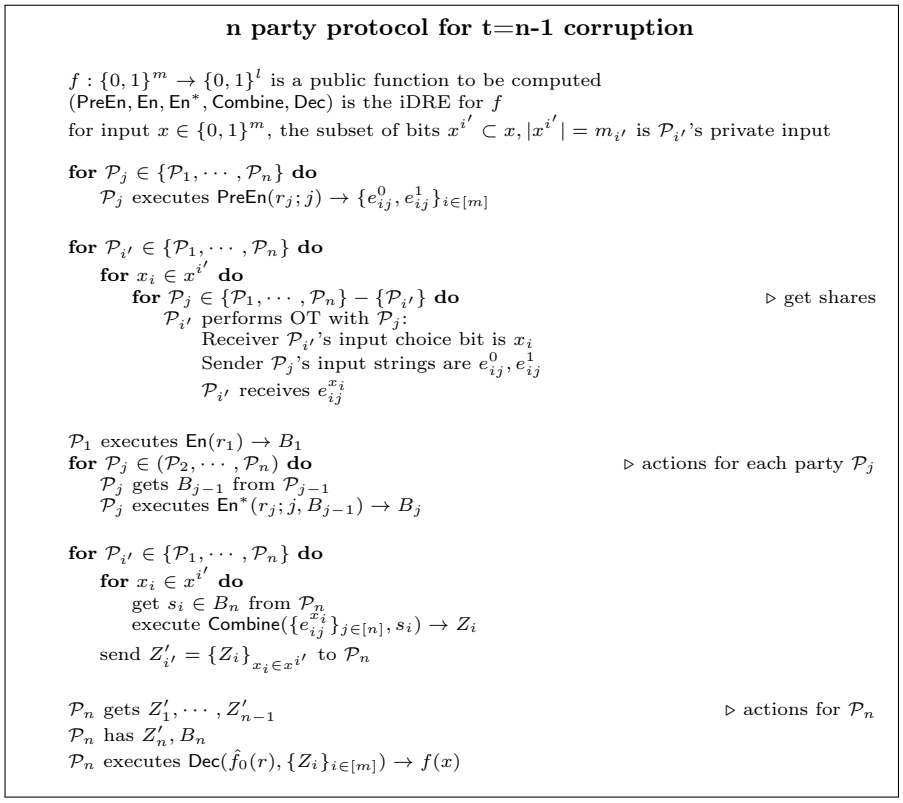

Fig. 3 Semi-honest MPC protocol based on iDRE

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.

AHKP22. Anasuya Acharya, Carmit Hazay, Vladimir Kolesnikov, and Manoj Prabhakaran. Scales: Mpc with small clients and larger ephemeral servers. *IACR Cryptol. ePrint Arch.*, page 751, 2022.

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.

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.