On-Line/Off-Line Leakage Resilient Secure Computation Protocols

Chaya Ganesh* Vipul Goyal† Satya Lokam‡

Abstract

We study the question of designing leakage-resilient secure computation protocols. Our model is that of *only computation leaks information with a leak-free input encoding phase*. In more detail, we assume an offline phase called the *input encoding phase* in which each party encodes its input in a specified format. This phase is assumed to be free of any leakage and may or may not depend upon the function that needs to be jointly computed by the parties. Then finally, we have a *secure computation phase* in which the parties exchange messages with each other. In this phase, the adversary gets access to a *leakage oracle* which allows it to download a function of the computation transcript produced by an honest party to compute the next outgoing message.

We present two main constructions of secure computation protocols in the above model. Our first construction is based only on the existence of (semi-honest) oblivious transfer. This construction employs an encoding phase which is dependent of the function to be computed (and the size of the encoded input is dependent on the size of the circuit of the function to be computed). Our second construction has an input encoding phase independent of the function to be computed. Hence in this construction, the parties can simple encode their input and store it as soon as it is received and then later on run secure computation for any function of their choice. Both of the above constructions, tolerate *complete leakage* in the secure computation phase.

Our second construction (with a function independent input encoding phase) makes use of a fully homomorphic encryption scheme. A natural question that arises is "can a leakage-resilient secure computation protocol with function independent input encoding be based on simpler and weaker primitives?". Towards that end, we show that any such construction would imply a secure two-party computation protocol with sub-linear communication complexity (in fact, communication complexity independent of the size of the function being computed).

Finally, we also show how to extend our constructions for the continual leakage case where there is: a one time leak-free input encoding phase, a leaky secure computation phase which could be run multiple times for different functionalities (but the same input vector), and, a leaky refresh phase after each secure computation phase where the input is "re-encoded".

*IIT Madras, chaya.ganesh@gmail.com. Work done in part while visiting Microsoft Research, India.

†Microsoft Research, India, vipul@microsoft.com.

‡Microsoft Research, India, satya@microsoft.com.

1 Introduction

Secure multi-party computation allows a set of n parties to compute a joint function of their inputs while keeping their inputs private. The first general solutions for the problem of secure computation were presented by Yao [Yao86] for the two-party case (with security against semi-honest adversaries) and Goldreich, Micali and Wigderson [GMW87] for the multi-party case (with security against malicious adversaries). These (and subsequent) protocols for secure computation were designed under the assumption that the local computations done by each of the parties on their private data are opaque.

In recent years, a vibrant area of research dealing with "leakage on the local computations" has emerged. This is a well motivated direction since sometimes, the local computations are not fully opaque and real world adversaries can exploit leakage from side channel attacks. There has been rapid progress on developing cryptographic primitives resilient against leakage of internal state in various models. There have been proposals of leakage resilient pseudorandom generators, public key encryption scheme, signature scheme, identity based encryption, etc (see [DP08, FKPR10, NS09] and the references therein). While significant progress has been made on designing cryptographic *primitives* secure in the presence of leakage, to our knowledge, there has not been any work on designing cryptographic *protocols* which tries to relax the assumption that the honest party machines are black-boxes with all internal computation hidden from the adversary.[1] In particular, while the protocol is running, what if the adversary manages to obtain a side channel allowing it to peek inside the honest party machines (in addition to completely corrupting some of the parties)? Can one still design secure computation protocols in this setting?

Our Results. We study the question of designing leakage-resilient secure computation protocols in this work. Our model is that of *only computation leaks information with a leak-free input encoding phase*. In more detail, our model has the following two phases.

We first assume an *offline phase* called the *input encoding phase*. This phase can be run in isolation and the parties need not be connected to the network. Hence, this phase is assumed to be free of any leakage. In this phase, each party encodes its input in a specified format. The encoding may or may not depend upon the function that needs to be jointly computed by the parties later on.

Then finally, we have a *secure computation phase* in which the parties exchange messages with each other. In this phase, the adversary gets access to a *leakage oracle* which allows it to download a function of the computation transcript produced by an honest party to compute the next outgoing message (this is, of course in addition to be able to completely corrupt a subset of the parties). If in any given round, the computation transcript "touches" only a subset of the bits of the encoded input (stored as a result of the input encoding phase), the leakage can depend only on that subset of the bits (i.e., only computation leaks information [MR04]).

Note that to hope to be able to have secure computation protocols, an input encoding phase is indeed *necessary*. This is because the security is violated even if a single bit of the initial input of an honest party is leaked to the adversary. Furthermore, if the adversary can later download a function of the entire encoded input, it can at least download a single bit of the initial input. Hence the assumption "only computation leaks information" also appears to be necessary. We emphasize that the security guarantees our protocols should satisfy correspond the standard ideal world (with no leakage allowed in the ideal world).

We present two main constructions of secure computation protocols in the above model. Our first construction is based only on the existence of (semi-honest) oblivious transfer. This construction employs an encoding phase which is dependent of the function to be computed (and the size of the encoded input is dependent on the size of the circuit of the function to be computed). Our second construction has an input encoding phase independent of the function to be computed. Hence in this construction, the parties can simple encode their input and store it as soon as it is received and then later on run secure computation for any function of their choice.

Both of the above constructions, somewhat surprisingly, tolerate *complete leakage* in the secure computation phase. That is, the adversary can fully observe the entire computation transcripts of the honest parties in the secure computation phase (including the bits of the encoded input that were used in running the protocol).

Our second construction (with function independent input encoding) makes use of a fully homomorphic encryption (FHE) scheme [Gen09] (see also [vDGHV10, Gen10]). A natural question that arises is "can a leakage-resilient

[1] Please see the end of this section for a discussion of the concurrent independent work.

secure computation protocol with function independent input encoding phase be based on simpler and weaker primitives?". In fact, can we even have a leakage-resilient secure computation protocol (based on weaker primitives) where the *size* of the encoded input is independent of the size of the circuit of the function to be computed? Towards that end, we show that any such construction would imply a secure two-party computation protocol with sub-linear communication complexity (in fact, communication complexity independent of the size of the function being computed). Note that constructing such a protocol was a central open problem in the field of secure computation (until a construction for FHE was proposed by Gentry [Gen09]). Currently, the only known way to construct a sub-linear communication complexity secure computation protocol is to rely on a FHE scheme.

Finally, we also show how to extend our constructions for the continual leakage case where there is: a one time leak-free input encoding phase, a leaky secure computation phase which could be run multiple times for different functionalities (but the same input vector), and, a leaky refresh phase after each secure computation phase where the input is "re-encoded". As before, the secure computation phase can tolerate complete leakage. However in the refresh phase, the leakage is bounded by a parameter t (which can be any apriori chosen polynomial in the security parameter).

Our Techniques. Our primary tool is to construct and store a number of garbled circuits in the input encoding phase and use them later on in the secure computation phase. For simplicity, we focus on the two-party case; similar ideas are applicable to the multi-party setting as well. The idea of our first construction is as follows. We compile any given (semi-honest) secure computation protocol Π into a leakage-resilient one as follows. In the input encoding phase, every party creates and stores a number of garbled circuits corresponding to the next message function of the underlying protocol. In the secure computation phase, the outgoing message of a party is computed as an output of one of the these garbled circuits. To evaluate such a garbled circuit, only the appropriate wire keys are "read" from the encoded input (one for each input wire of the garbled circuit). Now the computation transcript of the secure computation phase consists primarily of the transcript of evaluation of such garbled circuits (one for each round). However such evaluation transcripts can be simulated and hence the only valuable information that is revealed from leakage is the output of such a garbled circuit, which is just the next output message in the underlying protocol Π (there are caveats like a party needs to keep secret state between the rounds which can be dealt with using standard ideas). This idea is similar in spirit to the ones used to construct one-time programs [**?**]. In one-time programs, the security is based on the fact that only the appropriate wire keys are read from the given hardware tokens. Hence, one-time programs enable secure evaluation computation of a (non-interactive) function in the only-computation leaks information model.

The above idea "almost works" except for the following (rather subtle) problem. The real adversary $\mathcal{A}$ (given to us) expects to see the leakage along with every message it receives and then may *adaptively decide* the next outgoing message.

- To use the simulator S_{Π}, we need to construct an adversary $\mathcal{A}'$ for the protocol Π on which S_{Π} can be run.
- One natural high level idea is to construct another adversary $\mathcal{A}'$ for Π using the real adversary $\mathcal{A}$. To use $\mathcal{A}$, the new adversary $\mathcal{A}'$ will produce all the leakage to be given to $\mathcal{A}$ internally by constructing a *simulated* garbled circuit for each round. Hence now it seems S_{Π} can be simply run on $\mathcal{A}'$ (which is now a new adversary for the same protocol Π).
- The simulator S_{Π} of the underlying protocol Π may work by rewinding $\mathcal{A}'$. However once $\mathcal{A}'$ is rewound by S_{Π} in any given round, for that round, $\mathcal{A}'$ ends up giving to $\mathcal{A}$ *multiple* evaluation transcripts of the same simulated garbled circuit (on different inputs). However since we know how to prove the indistinguishability of a simulated garbled circuit from a real one only if the garbled circuit is evaluated *once*, our hybrid arguments completely break down. In more detail, once our simulator starts rewinding in the hybrid experiments, we can no longer rely on the indistinguishability of the real garbled circuit execution from a simulated one.[2]

[2]The problem of additional leakage while a simulator rewinds the adversary has also been observed in the context of leakage-resilient zero-knowledge [GS11]. In [GS11], such leakage in fact translated to additional leakage in the ideal world. For a comparison of our work to [GS11], please see the end of this Section.

- An initial idea to solve this problem is to have $\mathcal{A}'$ generate the simulated garbled circuit from scratch (to be given to $\mathcal{A}$) every time the protocol in rewound in a given round. However now since the machine $\mathcal{A}'$ needs to be aware of when it is being rewound and be allowed to change its random tape every time it is rewound, the success of the simulator S_Π is no longer guaranteed.

Our final idea to solve this problem continues to be to generate the simulated garbled circuit from scratch every time the protocol in rewound in a given round. However, this does not allow us to have a direct reduction to the security of the protocol Π (and might not work with a general rewinding strategy that S_Π may employ). Instead, we add a preamble to the protocol Π and push all the rewinding to performed to that preamble while the remaining simulation will be done in an straight-line manner.

For our second construction, we first use FHE for the evaluation of the next message function and hence end up with the next message to be sent in an encrypted form. We then use a garbled circuit (constructed in the input encoding phase) to get this next message decrypted. This is as opposed to directly using a garbled circuit to perform the entire computation. Hence the garbled circuit generated in the input encoding phase need only be able to decrypt ciphertexts of size dependent only the size of the messages in the protocol Π (and independent of the complexity of the next message function in particular). Finally, our underlying protocol Π has communication complexity (i.e., message sizes) independent of circuit size of the function to be computed (and dependent only the security parameter and the size of the input). Such a construction for Π can be obtained again by using FHE and Kilian's efficient PCP based zero-knowledge arguments [Kil95]. This construction also suffers from similar caveats relating to adversary choosing the next outgoing message adaptively on the leakage (and the solutions to these problems make use of the same ideas as in the previous construction).

Finally, we prove that any leakage-resilient secure computation protocol C_Π with function independent input encoding implies a secure two-party computation protocol Σ with computation complexity independent of the size of the circuit being computed. The main idea we use to construct such a low communication complexity protocol is for the protocol participant of Σ to internally emulate the parties of C_Π but partition the load of emulating such parties in a highly "unbalanced" way. In more detail, the second party in Σ does much of the work of emulating *both parties* of the protocol C_Π. This leads to the second party knowing about the internal state of both parties in the protocol C_Π; however here we can rely on the leakage-resilience of C_Π to argue security. The only communication is when the second party needs to read some bits of the encoded input of the first party in C_Π.

Concurrent Independent Work. Independent of our work, leakage resilient secure computation protocols have been proposed in the "plain" model where there is no leakage-free offline phase and the adversary may ask for leakage on the entire memory [GS11, DHP11, BCH11].[3] These works necessarily relax the security definition by allowing leakage on the input in the ideal world and hence are incomparable to ours.

2 Preliminaries and Model

2.1 Definition - Leakage Resilient Secure Computation

We model the leakage during the computation, by giving the adversary access to a leakage oracle.

Definition 1. ***Leakage Oracle:*** *Given a computation s (we assume s is given by a Boolean circuit) and λ, $0 \leq \lambda \leq 1$, a leakage oracle $\mathcal{O}_s^\lambda$ is defined as follows. For an input w, oracle $\mathcal{O}_s^\lambda(w)$ evaluates the queried PPT function on the transcript of the computation of s on w. Specifically, let $\tau(s, w)$ denote the transcript of computation of s on w, i.e., it consists of the input, output, and all the intermediate values in the circuit during the computation. Let h be any PPT* leakage *function whose output length is at most a λ-fraction of its input length. Now, on query h, $\mathcal{O}_s^\lambda(w)$ responds with $h(\tau(s, w))$. We call $\mathcal{O}_s^\lambda = \{\mathcal{O}_s^\lambda(w)\}_w$ the λ-bounded leakage oracle for s, where w ranges over all possible inputs to circuit s.*

When $\lambda = 1$, we call this the complete *leakage oracle for s and omit the superscript.*

We now give a formal definition of security against malicious adversaries in the presence of leakage in the ideal/real simulation paradigm. The execution in the ideal and real world are as follows:

[3]The paper [GS11] as well as an earlier draft of our work were both submitted to Crypto 2011 where [GS11] was accepted while our work was rejected.

Execution in the Ideal Model

Inputs: Each party obtains an input; the i^{th} party's input is denoted by x_i; we assume that the inputs are of the same length n. The adversary $\mathcal{A}$ receives an auxiliary-input z.
Send inputs to trusted party: The honest party P_j sends its received input x_j to the trusted party. A malicious party may depending on its input (as well as on its coin tosses and the auxiliary input) either abort or send some other input of the same length to the trusted party.
Trusted party answers the first party: In case the trusted party received the input pair, (u_1, u_2), the trusted party first replies to the first party with $f(u_1, u_2)$.
Trusted party answers the second party: In case the first party is malicious, it may depending on its input and the trusted party's answer, decide to abort the trusted party. In this case, the trusted party sends abort to the second party. Otherwise, the trusted party sends $f(u_1, u_2)$ to the second party.
Outputs: An honest party always outputs the message it obtained from the trusted party. A malicious party may output any arbitrary (probabilistic polynomial-time computable) function of the initial input and the message obtained from the trusted party.

Definition 2. *The output of honest party and the adversary in an execution of the above model is denoted by* $IDEAL_{f,\mathcal{A}(z)}(x_1, x_2)$.

Execution in the Real Model

Let P_1 and P_2 be two parties trying to securely compute functionality $f : (\{0,1\}^*)^2 \to (\{0,1\}^*)^2$. W.l.o.g., let P_1 be the corrupt party (P_2 is honest). We want to model leakage from the honest party P_2. Note that we only need to consider leakage from honest parties. Let $\mathcal{A}$ be a PPT adversary controlling malicious party P_1. The adversary $\mathcal{A}$ can deviate from the protocol in an arbitrary way. In particular, $\mathcal{A}$ can access leakage from P_2 via its leakage oracle throughout the execution of the protocol. In addition, $\mathcal{A}$'s behavior may depend on some auxiliary input z. More details follow.

Our real world execution proceeds in two phases: a leakage-free *Encoding phase* and a *Secure Computation Phase* in which leakage may occur. In the Encoding phase, each party P_i, locally computes an encoding of its input and any other auxiliary information. The adversary has no access to the leakage oracle during the encoding phase. In the Secure Computation phase, the parties begin to interact, and the adversary gets access to the leakage from the local computation of P_2. Specifically, let s_2 be a circuit describing the internal computation of P_2. The circuit s_2 takes as input the history of protocol execution of the *secure computation* phase so far, and some specified bits of the encoded input, and outputs the next message from P_2 and the new internal state of P_2. Thus, by Definition 1, the leakage is accessible to the adversary via the oracle $\mathcal{O}_{s_2}$, which we abbreviate by $\mathcal{O}_2$. Thus, at the end of round j of the protocol, $\mathcal{A}$ can query $\mathcal{O}_2$ for any PPT function h_j applied on the transcript of the jth round computation of the honest party P_2. We denote this leakage information by l_2^j.

We consider two variants of the above model: one in which the *Encoding phase* can depend on the function to be computed, and another stronger model in which the leakage-free Encoding phase is independent of the function.

Definition 3. *We define* $REAL^{\mathcal{O}^\lambda}_{\pi,\mathcal{A}(z)}(x_1, x_2, \kappa)$ *to be the output pair of the honest party and the adversary* $\mathcal{A}$ *from the real execution of* π *as defined above on inputs* (x_1, x_2)*, auxiliary input* z *to* $\mathcal{A}$*, with oracle access to* $\mathcal{O}_2^\lambda$*, and security parameter* κ*.*

Definition of Security

Definition 4. *Let* f, π*, be as described above. Protocol* π *is said to* securely compute f in the presence of $\lambda-$leakage *if for every non-uniform probabilistic polynomial-time pair of algorithms* $\mathcal{A} = (A_1, A_2)$ *for the real model, there exists a non-uniform probabilistic polynomial-time pair* $S = (S_1, S_2)$ *for the ideal model such that*

$$IDEAL_{f,S(z)}(x_1, x_2) \stackrel{c}{\equiv} REAL^{\mathcal{O}^\lambda}_{\pi,\mathcal{A}(z)}(x_1, x_2, \kappa)$$

where $x_1, x_2, z \in \{0,1\}^*$*, such that* $|x_1| = |x_2|$ *and* $|z| = \mathsf{poly}(|x_1|)$*. When* $\lambda = 1$*, protocol* π *is said to* securely compute f in the presence of complete leakage.

The model for the multi-party setting is analogous to one given above for the two-party case. For lack of space, the details are provided in the Appendix A.

3 The Basic Construction

We give a compiler that transforms any semi-honest secure multiparty protocol into a leakage resilient multiparty protocol secure against malicious parties and resilient against complete leakage ($\lambda = 1$) from honest parties. For simplicity, we describe our results for the two party case. They can be naturally extended to the multiparty case using known techniques. We sketch such an extension in Section C.2.

The high level idea of the compiler is to garble the "Next Message" functions of a semi-honest two party computation protocol in a leakage-free preprocessing phase. The internal states of the parties are maintained in an encrypted form using a semantically secure public key encryption scheme. The private input of each party is included as part of that party's initial (encrypted) state. All updates to the internal states are also performed on the corresponding ciphertexts. The garbled circuit of the Next Message function of a party acts on its encrypted internal state and the message from the other party to produce its new (encrypted) state and the message to be sent to the other party.

We start with any protocol Π secure against semi-honest adversaries, which is converted into an intermediate protocol Π_{INT}, and then compiled by the compiler C. The compiled protocol C_{Π} is secure against malicious adversaries in the presence of complete leakage.

3.1 Protocol Π_{INT}

Let P_1 and P_2 be two parties with private inputs x_1 and x_2, respectively, where $x_i \in \{0,1\}^n$. κ is the security parameter. Let Π be a given semi-honest secure protocol. The protocol Π_{INT} proceeds in two phases:

Phase I. In this phase, each party P_i commits to its input and randomness. Each party also engages in a challenge-response step with the other party.

• $P1$ does the the following:

1. Let $X_1 = (x_1, r_1)$, where r_1 is P_1's randomness to be used for executing the underlying protocol Π.
   P_1 breaks the string X_1 into k pairs of 2-out-of-2 secret shares $(v_1^0, v_1^1), \ldots, (v_k^0, v_k^1)$, where $v_i^0 \oplus v_i^1 = X_1$, and k is $\mathsf{poly}(\kappa)$.
   P_1 creates commitments to all k pairs of strings using a (non-interactive)[4] statistically binding commitment scheme COM and sends them to P_2.

2. P_1 and P_2 now engage in a challenge-response interaction in which P_1 opens one commitment in each pair as per the challenge string:
   P_2 sends a random k bit challenge $e = (e_1, \ldots, e_k)$.
   P_1 opens the commitments to $v_1^{e_1} \ldots v_k^{e_k}$.

3. P_2 checks if the openings are valid. If cheating is detected, P_2 aborts.

4. P_2 acts symmetrically. That is, it commits to, and engages in a challenge-response interaction with P_1 for its input and randomness $X_2 = (x_2, r_2)$;

5. P_1 picks a random string R_1, where $|R_1| = |X_2|$. P_1 now also commits to R_1 in an extractable manner as above. That is, by breaking R_1 into k pairs of secret shares, receiving a challenge string from P_2 and opening the commitments for the shares demanded by the challenge string. P_2 acts symmetrically by committing to a random string R_2 in an extractable manner.

At the end of this phase, party P_i is committed to its input and randomness X_i, and the random string R_i.

Phase II. In this phase an execution of the semi-honest protocol Π is run. First, a coin flipping protocol is run to force the randomness to be unbiased, and then Π is compiled using witness indistinguishable arguments (WI) to enforce honest behaviour on the parties.

[4] We use a non-interactive commitment scheme only for simplicity of presentation, our construction can also be instantiated with an interactive statistically binding commitment scheme.

1. *Coin tossing*: P_1 and P_2 execute a coin flipping protocol.
   (a) P_1 chooses a uniformly random string ρ_2, and sends it to P_2.
   (b) P_2 chooses a uniformly random string ρ_1, and sends it to P_1.
   (c) P_1 defines $r_1'' = r_1 \oplus \rho_1$, and P_2 defines $r_2'' = r_2 \oplus \rho_2$. r_1'' and r_2'' are the random coins that P_1 and P_2 will use in the execution of protocol Π.

2. *Run Protocol* Π: Let T_i^j denote the transcript (history of communication and internal computation) of messages of protocol Π before P_i is supposed to send its message M_i^j in round j. Each message sent by Party P_i in Π is compiled into a message block in Π_{INT}.
   - P_1 sends the next message M_1^j as per protocol Π. Now, P_1 and P_2 engage in the execution of a WI-argument, where P_1 proves the following statement:

   EITHER (a) (i) there exists $X_1 = (x_1, r_1)$, such that the shares committed to in phase 1 can be decommitted to shares of X_1, and (ii) the sent message M_1^j is consistent with input x_1 and randomness r_1'', that is, $M_1^j = \Pi(T_1^j, x_1, r_1'')$,
   OR (b) the shares of the random string R_1 committed to in step (4) of phase 1, are shares of X_2.

   P_1 uses the witness corresponding to the part (a) of the statement above.
   - P_2 acts symmetrically.

The protocol Π_{INT} contains several instances of WI-arguments. The proof statement for each WI instance consists of two parts. In an actual execution of the protocol, the parties use the witness to the first part of the proof statement.

3.2 The Compiler

- **Input**: P_1 has input $x_1 \in \{0,1\}^n$ and P_2 has input $x_2 \in \{0,1\}^n$.
- **Encoding phase**: This is done in a leakage-free setting. For concreteness, we assume that in the original protocol Π, P_1 sends the first message. We use the notation m_i^j to denote the jth message sent by party i. Thus jth message from party P_1 followed by jth message from party P_2 constitutes the jth round.

  Party P_1 does the following:

  1. Initialize the secret state st_1^0 with the private input x_1 and private random tape for the protocol Π_{INT}.
  2. Choose a public key and secret key pair (pk_1, sk_1) of an Encryption scheme E.
     $(pk_1, sk_1) \leftarrow KeyGen(1^\kappa)$
  3. Encrypt the initial secret state st_1^0 under pk_1.
     $E[st_1^0] \leftarrow Encrypt(pk_1, st_1^0)$
  4. Let $m_1^j \leftarrow \mathrm{NextMsg}_1^j(m_2^{j-1}, st_1^{j-1})$ be the Next Message function of Π_{INT} to compute the jth message m_1^j that is to be sent to P_2. Let $NextMsgC_1^j$ be the circuit with the following functionality. It has the keys pk_1 and sk_1 hardcoded and takes as input $m_2^{j-1}, E[st_1^{j-1}]$. It decrypts $E[st_1^{j-1}]$, executes $\mathrm{NextMsg}_1^j(m_2^{j-1}, st_1^{j-1})$, and hence computes the next message of P_1 as m_1^j. It outputs this next message and the new encrypted state $(E[st_1^j], m_1^j)$ and halts.
     $E[st_1^j], m_1^j \leftarrow NextMsgC_1^j(m_2^{j-1}, E[st_1^{j-1}])$.
  5. For every round j, using the garbled circuit construction of Yao, garble the circuit $NextMsgC_1^j(m_2^{j-1}, E[st_1^{j-1}])$ to get the garbled circuit $\mathrm{NextMsgGC}_1^j$.

  P_2 acts symmetrically.

- **Secure Computation phase**: This runs the leakage resilient protocol $C_{\Pi}(x_1, x_2)$ defined below.

  The parties emulate the underlying protocol Π_{INT}, by replacing every call to the Next Message function by an invocation of the corresponding garbled circuit computed during the preprocessing phase.

  Party P_1 does the following:

  1. In round j, evaluate the garbled circuit $\mathrm{NextMsgGC}_1^j$ under the wire keys corresponding to the input $m_2^{j-1}, E[st_1^{j-1}]$. That is, from the encoded input, only the wire keys corresponding to the input $m_2^{j-1}, E[st_1^{j-1}]$ are "touched".

  $$E[st_1^j], m_1^j \leftarrow \mathrm{NextMsgGC}_1^j(m_2^{j-1}, E[st_1^{j-1}])$$

  2. Send m_1^j to the other party.
  3. Update the secret state with $E[st_1^j]$ and wait for the next message from P_2 if it exists; otherwise halt.

  P_2 acts symmetrically.

4 Proof of Security

Theorem 1. *Assuming Π securely computes f in the semi-honest model, the protocol C_{Π} securely computes f as in Definition 4 for $\lambda = 1$.*

The proof constructs a simulator (ideal world adversary) whose output is computationally indistinguishable from the view of the real world adversary in an actual run of the protocol. The simulator will access the real world adversary and a trusted party.

4.1 Description of the Simulator

W.l.o.g., we assume P_1 is corrupt. Let A be the adversary controlling the corrupt party. We describe the simulator S.

Phase I.

- In this phase, the simulator S plays honest receiver, and obtains from A a transcript τ of the commit-challenge-response phase for the adversary's input and randomness.
- If τ is not an accepting transcript, then S outputs $\perp$, and aborts. If the transcript is accepting with challenge string e, then S rewinds A to the point where A expects a challenge string, i.e., step (2) of Phase I. It keeps rewinding until another accepting transcript is received, with another challenge e'. If $e = e'$, S outputs $\perp$, and aborts. Otherwise, from the responses of A to two distinct challenges e, e', the appropriate shares are combined and the adversary's input and randomness X_1' is recovered.
- S now plays honest committer and commits to a random string X_2' and responds to the challenge honestly.
- S receives commitments to a random string R_1'. S now sends commitments to random shares of the extracted X_1' to A (instead of random string R_2' of the honest party), and honestly responds to the challenge.

Phase II.

- Let S_{Π} be the simulator for the semi-honest protocol Π. S runs the simulator S_{Π} on the adversary's input X_1' that S extracted in phase I. S_{Π} outputs a transcript $T = (M_1^1, M_2^1, \ldots, M_1^r, M_2^r)$ of the execution of Π, and an associated random string r_A.
- Now, S computes a random string, $r = r_A \oplus r_1'$. where r_1' is the randomness S extracted from A in phase I (as part of X_1'). S now forces A to use the randomness r_A in the execution of Π. To do this, during the coin flipping phase, S sends r to A.

- Now, simulator S forces the transcript T on A during the execution of Π. In round j, S sends the message M_2^j to A, instead of a message according to the input and randomness it committed to in phase I. In the associated WI, S uses the witness corresponding to the second part of the statement, that is the shares of the random string it committed to in phase 1, are shares of X_1'. The reply of A must be M_1^{j+1}, that is it must be according to the transcript T, except with negligible probability. If the reply of A is not according to T and yet A succeeds in the associated WI, then S outputs $\perp$ and aborts.
- The simulator now picks a key pair (pk_1, sk_1) of the Encryption scheme E. Let $E[M]$ denote the encryption of a message M under pk_1.
- Throughout, with every message M_2^j that S sends to the adversary, it also sends the associated leakage l_2^j which S computes in the following way: In round j, S picks a random string $rand_j$. It then constructs a fake garbled circuit of the Next Message function of Π_{INT} that always outputs M_2^j and $E[rand_j]$. S evaluates this fake garbled circuit under the wire keys corresponding to inputs M_1^{j-1} and $E[rand_{j-1}]$, and sets l_2^j to the transcript of this evaluation. The *evaluation transcript* consists of garbled gate tables of every gate in the circuit, output decryption table, and a single garbled value for every wire in the circuit. **Whenever S rewinds A (S rewinds A only in phase I) and is required to give leakage for the same round again, it constructs a fresh fake garbled circuit** as opposed to giving the evaluation transcript of the same one on a different input.

Running time of S: It is easy to see that S either aborts (with negligible probability) or rewinds an expected constant number of times. It follows that S runs in expected polynomial time.

We prove the indistinguishability of the views going from the real protocol to the simulated one through a series of hybrids. We prove that the real protocol view is indistinguishable from that in the simulated execution by proving indistinguishability between every pair of successive hybrids. The detailed proof appears in Appendix C.1.

An extension to the multi-party case can be found in Appendix C.2.

5 Construction with Function Independent Encoding

We now give a construction of another compiler such that the input encoding phase is independent of the function f to be computed. To do this, we start with a generic protocol Π with round complexity and communication complexity independent of the function to be computed. We then use a compiler C in a similar way as before to transform Π into a protocol C_Π that is secure against a malicious adversary in the presence of complete leakage. Recall that the secure computation phase is subject to complete leakage, i.e., all the data and internal states of local computations in the online phase are completely visible to the adversary, whereas the input encoding phase is leakage-free.

5.1 The Generic Protocol Π

Our generic two party computation protocol Π, which is secure against semi-honest adversaries, is constructed based on a Fully Homomorphic Encryption (FHE) scheme E.

1. **Input:** P_1 has input $x_1 \in \{0,1\}^n$ and P_2 has input $x_2 \in \{0,1\}^n$
2. P_1 generates a key pair for FHE scheme E and sends the public key to P_2.
   $(pk, sk) \leftarrow KeyGen(1^\kappa)$
3. P_1 encrypts her input and sends the ciphertext to P_2.
   $E[x_1] \leftarrow Encrypt(x_1, pk)$
4. P_2 encrypts his input and evaluates the circuit $\mathcal{C}$ homomorphically.
   $E[f(x_1, x_2)] \leftarrow Eval(\mathcal{C}, E[x_1], E[x_2], pk)$
   P_2 sends $E[f(x_1, x_2)]$ to P_1.
5. P_1 decrypts to obtain $f(x_1, x_2)$. She sends $f(x_1, x_2)$ to P_2.

It is clear that the communication complexity of the above protocol Π depends only on the size of the input n and the security parameter κ and is independent of the function f (in particular, independent of the size of the circuit $\mathcal{C}$).

5.2 The Intermediate Protocol Π_{INT}

The protocol Π above is converted into an intermediate protocol Π_{INT}, along similar lines as described in Section 3.1, *except* that we use efficient WI arguments *based on PCP* in step (2) of Phase II. The communication complexity of PCP based WI-arguments is a polynomial only in the security parameter [Kil95]. Therefore the communication complexity of Π_{INT} is also independent of f and the round complexity is a constant. Note, however, that the Next Message function of Π_{INT} depends on the circuit $\mathcal{C}$ computing f.

5.3 Compiler C

Our compiler starts with the protocol Π_{INT} described above and produces the compiled protocol C_{Π}.

- **Input Encoding phase:** This phase is independent of the function to be computed.

  1. Party i chooses a key pair (pk_i, sk_i) of an FHE $E = (KeyGen, Encrypt, Decrypt, Eval)$.
  $$(pk_i, sk_i) \leftarrow KeyGen(1^{\kappa})$$
  2. The initial internal state st_i^0 of Party i is initialized with the private input and randomness of P_i for the execution of Π_{INT}. It is then encrypted under the chosen key pk_i.
  $$E[st_i^0] \leftarrow Encrypt(pk_i, st_i^0)$$
  3. Let $Decrypt(sk_i, c)$ be the decryption circuit of the FHE scheme for Party i. Using the Garbled circuit construction of Yao, garble the decryption circuit to get $DecGC_i$. Let t be the number of rounds in the protocol Π_{INT}. Construct t independent garbled circuits of $Decrypt(sk_i, c)$. Let $DecGC_i^j$ denote the garbled decryption circuit of Party i to be used in round j. The length of the ciphertext that the garbled circuit should handle as the input is the length of the messages in Π_{INT}. Thus it is polynomial in the security parameter and the input size and independent of f.

- **Secure Computation phase:**
  P_1 does the following:

  1. *Encryption:* Let m_2^{j-1} be the message received. Party P_1 encrypts the received message under its own public key.
  $$E[m_2^{j-1}] \leftarrow Encrypt(pk_1, m_2^{j-1})$$
  2. *Homomorphic evaluation:* Let $NextMsg_1^j$ be the circuit description of the next message function used by Party P_1 in round j in protocol Π_{INT}. This circuit is evaluated homomorphically to get an encryption of m_1^j in round j.
  $$(E[m_1^j], E[st_1^j]) \leftarrow Eval(E[st_1^{j-1}], E[m_2^{j-1}], NextMsg_1^j)$$
  3. *Garbled circuit evaluation:* Party 1 decrypts $E[m_1^j]$ by evaluating the Garbled circuit $DecGC_1^j$ under the wire keys corresponding to input $E[m_1^j]$.
  $m_1^j \leftarrow DecGC_1^j(sk_1, E[m_1^j])$
  4. Party P_1 sends m_1^j to P_2.
  5. Party P_1 updates the secret state to $E[st_1^j]$.

  Party P_2 acts symmetrically.

The security guarantees of the protocol C_{Π} are stated in the following theorem.

Theorem 2. *The protocol C_{Π} is secure against complete leakage, as in Definition 4,* with an encoding phase independent of the function to be computed.

The proof of this theorem follows from ideas similar to that in theorem 1 and can be found in appendix 2.

5.4 Extension to Multiparty case

We now sketch an extension of C_Π to the multi-party case. The underlying semi-honest secure protocol Π which we compile is as follows.

- The parties run a multi-party protocol for the following functionality: The functionality takes as input, the vector of inputs of all the parties. It generates the key pair for a fully homomorphic encryption scheme E. The functionality encrypts the input of each party and generates n-out-of-n secret shares of the decryption key. Each party gets as output, encryption of the inputs of all parties, the public key, and a share each of the secret key.
- The first party now evaluates the circuit homomorphically to get the encrypted output.

$$E[C(x_1, \ldots, x_n)] \leftarrow Eval(E[x_1], \ldots, E[x_n], C)$$

- The parties now run a multi-party protocol for the following functionality. The functionality takes as input the secret shares of the decryption key of each party, and outputs the decrypted output.

The above protocol is compiled as in Section 3.1 to get a two phase protocol Π_{INT} with WI arguments to enforce honest behavior. Protocol Π_{INT} is compiled with compiler C as discussed in 5.3 to get a leakage resilient multi-party protocol C_Π.

6 Is FHE Necessary for a Construction with Function Independent Encoding?

Our protocol C_Π in the previous section uses Fully Homomorphic Encryption as a component to realize function-independent encoding. In this section, we address the question if this is necessary. We show that leakage resilience with function independent encoding implies two-party secure computation with a communication complexity that is independent of the function to be computed. In particular, given a leakage resilient protocol C_Π that securely computes f in the presence of complete leakage and uses a function-independent encoding, we construct a secure two party computation protocol Σ whose communication complexity is independent of f. Realizing the latter task was a major open question and is currently known to be possible only if one uses FHE. Thus, being able to avoid FHE in building a leakage resilient protocol with function-independent encoding would have an alternative solution to this open question that does not rely on FHE.

In what follows, we denote by P_1 and P_2 the players in the leakage resilient protocol C_Π and by p_1 and p_2 the players in the communication-efficient protocol Σ. Given C_Π we show how to build Σ. The protocol Σ is secure against semi-honest adversaries which do not deviate from the protocol, but they may only try to get more information than they are authorized to. This can be extended to the malicious case by compiling Σ using PCP-based efficient zero knowledge arguments to prove honest behavior ([GMW87], [Kil95]). (The communication complexity of Kilian's argument system is independent of the complexity of verifying the statement in question.)

Protocol Σ: Parties p_1 and p_2 with respective inputs x_1 and x_2 wish to jointly compute $f(x_1, x_2)$.

- *Input Encoding phase:* The parties encode their input (and random tape) locally exactly as per the instructions of the protocol C_Π (see encoding phase of section 5.3). Let m be the size of the encoded input.
- *Secure Computation Phase:* Party p_2 runs the programs of *both* parties P_1 and P_2 in protocol C_Π. That is, p_2 internally runs the parties P_1 and P_2 of C_Π. Whenever party p_2 needs to read a bit of the encoded input of Party 1, it queries p_1 with the index, and gets the encoded input bit. Party p_2 computes the output $f(x_1, x_2)$, and sends the output to party p_1.

Security of Σ: Here we only provide a proof sketch of the security of the protocol Σ. The details are straightforward.

Party p_1 is corrupt: We first consider the case when p_1 is corrupt. All p_1 learns in the execution of Σ is the indices queried by p_2. We observe that a dishonest P_1 learns this *and the leakage* in the protocol C_Π. Thus, the view of dishonest p_1 in Σ is a strict subset of the view of dishonest P_1 in C_Π. By the security of C_Π, protocol Σ is secure.

Party p_2 is corrupt: In the case when p_2 is dishonest, the view of p_2 consists of the entire view of P_2 in C_Π and the view of P_1 in C_Π in the secure computation phase. This is exactly the same as the view of a dishonest P_2 in the presence of the leakage oracle. By the leakage resilience of C_Π, protocol Σ is secure when p_2 is dishonest.

Thus, we conclude that Σ is secure two-party computation protocol.

Communication complexity of Σ: The communication complexity is the maximum number of queries from p_2 to p_1. If the size of the encoded input is m, then the communication complexity of the protocol Σ is $O(m \log m)$, independent of the size of the circuit being evaluated for f. Even the protocol secure against malicious adversaries has communication complexity independent of f, due to Kilian's PCP based construction of zero knowledge argument. Thus, we have

Theorem 3. *For two party computation, if there exists a leakage-resilient secure protocol with function-independent encoding, then there exists a secure protocol (against malicious adversaries) with communication complexity independent of the function being computed.*

7 Continual Leakage Resilient Protocol

We now construct a protocol which runs the input encoding phase just once, and runs the computation phase of the protocol more than once without having to run the leak-free phase again.

The Model. The ideal and the real model remain similar to the one time computation case (see Section 2). The main difference in the ideal world is that now there are k interactions: the parties interact k times, and compute a different functionality of the same inputs x_i. That is $f_j(x_1, x_2)$ is the functionality computed in the jth interaction with the trusted party. In the real model, apart from the leakage-free encoding phase and a leaky secure computation phase, there is a *refresh* phase as well. The adversary is allowed bounded leakage (of up to t bits) in *each* refresh phase. Between every secure computation phase, there is a refresh phase. However the leakage free encoding phase is only one time. More details are provided in the appendix E.

Our compiler starts with the protocol Π_{INT} as described in section 3.1 and produces the compiled protocol C_Π.

- **Input Encoding phase:** This phase is independent of the function to be computed. The first three steps of this phase remain the same as before.

  1. Party i chooses a key pair (pk_i, sk_i) of an FHE $E = (KeyGen, Encrypt, Decrypt, Eval)$.

  $$(pk_i, sk_i) \leftarrow KeyGen(1^\kappa)$$

  2. The initial internal state st_i^0 of Party i is initialized with the private input and randomness of P_i for the execution of Π_{INT}. It is then encrypted under the chosen key pk_i.

  $$E[st_i^0] \leftarrow Encrypt(pk_i, st_i^0)$$

  3. Let $Decrypt(sk_i, c)$ be the decryption circuit of the FHE scheme for Party i. Using the Garbled circuit construction of Yao, garble the decryption circuit to get $DecGC_i$. Let r be the number of rounds in the protocol Π_{INT}. Construct r independent garbled circuits of $Decrypt(sk_i, c)$. Let $DecGC_i^j$ denote the garbled decryption circuit of Party i to be used in round j. The length of the ciphertext that the garbled circuit should handle as the input is the length of the messages in Π_{INT}. Thus it is polynomial in the security parameter and the input size and independent of f.
  4. **Virtual Player Initialization.** Each party i initializes $3t+1$ virtual players (where t is the leakage bound of the refresh step). The virtual player j holds a share $sk_i[j]$ of the secret key sk_i as input. In other words, at this stage, the party simply divides the secret key sk_i into $3t+1$ shares $\{sk_i[1], \ldots, sk_i[3t+1]\}$ (using an additive secret sharing scheme) and stores these shares.

- **Secure Computation phase:**
  This phase remains the same as before. P_1 does the following:

1. *Encryption:* Let m_2^{j-1} be the message received. Party P_1 encrypts the received message under its own public key.
$$E[m_2^{j-1}] \leftarrow Encrypt(pk_1, m_2^{j-1})$$
2. *Homomorphic evaluation:* Let $NextMsg_1^j$ be the circuit description of the next message function used by Party P_1 in round j in protocol Π_{INT}. This circuit is evaluated homomorphically to get an encryption of m_1^j in round j.
$$(E[m_1^j], E[st_1^j]) \leftarrow Eval(E[st_1^{j-1}], E[m_2^{j-1}], NextMsg_1^j)$$
3. *Garbled circuit evaluation:* Party 1 decrypts $E[m_1^j]$ by evaluating the Garbled circuit $DecGC_1^j$ under the wire keys corresponding to input $E[m_1^j]$.
$$m_1^j \leftarrow DecGC_1^j(sk_1, E[m_1^j])$$
4. Party P_1 sends m_1^j to P_2.
5. Party P_1 updates the secret state to $E[st_1^j]$.

Party P_2 acts symmetrically.

- **Refresh Phase:**
Let $\mathcal{F}$ be the following $3t+1$-party (randomized) functionality:
  - It takes as input $sk_i[j]$ from player j for $j \in [3t+1]$. It reconstructs the secret key sk_i.
  - It computes $3t+1$ shares of sk_i using fresh randomness (using an additive secret sharing scheme as before). Denote the j-th share by $sk_i'[j]$. It gives $sk_i'[j]$ as output to the j-th player.
  - It also computes, for every round, a garbled circuit for decryption of a ciphertext using sk_i (as constructed in the leakage-free encoding phase). More precisely, it constructs r independent garbled circuits for $Decrypt(sk_i, c)$ and outputs that to each player.

  Party P_1 does the following. It internally runs the BGW protocol (guaranteing security for semi-honest players) [BOGW88] among the $3t+1$ players. The players hold inputs $sk_i[1], \ldots, sk_i[3t+1]$ and run the BGW protocol for the functionality $\mathcal{F}$. *The internal computation of each player is modeled as a separate sub-computation.* Hence, the adversary is allowed to ask for leakage individually on each of the $3t+1$ subcomputation as well as on the protocol transcript generated by the (virtual) interaction.

  Note that The j virtual players started with shares $sk_i[1], \ldots, sk_i[3t+1]$ and ended with new shares $sk_i'[1], \ldots, sk_i'[3t+1]$ using which the refresh phase can be run again. Furthermore, at the end of Refresh phase, P_1 has r independent garbled circuits of $Decrypt(sk_i, c)$ (obtained as output by each player), and hence the Secure Computation phase can be run again.

Sketch of Proof of security The security guarantees of the protocol C_Π are stated in the following theorem:

Theorem 4. *The protocol C_Π is secure against λ-continual leakage as in Definition 9 tolerating complete leakage in the secure computation phase, and t bits of leakage in the refresh phase, where t can be any apriori fixed polynomial in the security parameter κ.*

We construct a simulator and prove that the output is computationally indistinguishable from the view of a real world adversary in an actual run of the protocol. The simulator proceeds by simulating the Secure computation and the Refresh phase of each execution.
Simulation in the Secure Computation phase: The description of the simulator in this phase is the same as described in section 2.
Simulation during the Refresh phase: Since the adversary is restricted to t-bits of leakage, it can request leakage from at most t of the $3t+1$ subcomputations. Invoke the BGW simulator to simulate the view of the adversary by corrupting these (at most) t players and constructing their view.

The indistinguishability of the views follows from theorem 2 and the security of the BGW construction [BOGW88].

Acknowledgements. We wish to thank Yuval Ishai, Abhishek Jain, Amit Sahai and Daniel Wichs for useful discussions.

References

[BCH11] Nir Bitansky, Ran Canetti, and Shai Halevi. Leakage tolerant interactive protocols. Cryptology ePrint Archive, Report 2011/204, 2011.

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10. ACM, 1988.

[DHP11] Ivan Damgaard, Carmit Hazay, and Arpita Patra. Leakage resilient secure two-party computation. Cryptology ePrint Archive, Report 2011/256, 2011.

[DP08] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302. IEEE Computer Society, 2008.

[FKPR10] Sebastian Faust, Eike Kiltz, Krzysztof Pietrzak, and Guy N. Rothblum. Leakage-resilient signatures. In Daniele Micciancio, editor, *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 343–360. Springer, 2010.

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

[Gen10] Craig Gentry. Toward basing fully homomorphic encryption on worst-case hardness. In Tal Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 116–137. Springer, 2010.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. *In proceedings of 19th Annual ACM Symposium on Theory of Computing*, pages 218–229, 1987.

[Gol01] Oded Goldreich. *Foundations of Cryptography, Volume I: Basic Techniques*. Cambridge University Press, 2001.

[GS11] Sanjam Garg, Abhishek Jain 0002, and Amit Sahai. Leakage-resilient zero knowledge. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 297–315. Springer, 2011.

[Kil95] Joe Kilian. Improved efficient arguments. In *CRYPTO, volume 963 of LNCS*, pages 311–324, 1995.

[LP04] Yehuda Lindell and Benny Pinkas. A proof of yao's protocol for secure two-party computation. Cryptology ePrint Archive, Report 2004/175, 2004.

[MR04] S. Micali and L. Reyzin. Physically observable cryptography. In *TCC 2004*, pages 278–296, 2004.

[NS09] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 18–35. Springer, 2009.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In Henri Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2010.

[Yao86] A. C. Yao. How to generate and exchange secrets. In *FOCS '86: Proceedings of 27th Annual Symposium on Foundations of Computer Science*, pages 162–167, 1986.

A Definition for the Multi-party Case

We now give a formal definition of security against malicious adversaries in the presence of leakage in the ideal/real simulation paradigm for the multi-party case. The execution in the ideal and real world are as follows:

Execution in the Ideal Model .

Inputs: Each party obtains an input; the i^{th} party's input is denoted by x_i; we assume that the inputs are of the same length n. The adversary receives an auxiliary-input z.

Send inputs to trusted party: Any honest party P_j sends its received input x_j to the trusted party. A malicious party may depending on its input (as well as on its coin tosses and the auxiliary input) either abort or send some other input of the same length to the trusted party. This decision is made by $\mathcal{A}$ and may depend on x_i for $i \in \mathcal{I}$ and the auxiliary input z. Denote the vector of inputs sent to the trusted party by $\bar{w}$.

Trusted party answers adversary: The trusted party computes $(f_1(\bar{w}), \ldots, f_m(\bar{w}))$ and sends $f_i(\bar{w})$ to $\mathcal{A}$, for all $i \in \mathcal{I}$.

Trusted party answers honest parties: After receiving its outputs, the adversary sends either abort_i for some $i \in \mathcal{I}$ or $\mathsf{continue}$ to the trusted party. If the trusted party receives $\mathsf{continue}$ then it sends $f_i(\bar{w})$ to all honest parties $P_j (j \notin \mathcal{I})$. Otherwise, if it receives abort_i for some $i \in \mathcal{I}$, it sends abort_i to all honest parties.

Outputs: An honest party always outputs the messages it obtained from the trusted party. The corrupted parties output nothing. The adversary $\mathcal{A}$ outputs any arbitrary (probabilistic polynomial-time computable) function of the initial inputs $\{x_i\}_{i \in \mathcal{I}}$ and messages obtained from the trusted party.

The output of honest parties and the adversary in an execution of the above model is denoted by $IDEAL_{f,A(z),\mathcal{I}}(\bar{x}, \kappa)$ where κ is the security parameter.

Execution in the Real Model .

Let the set of parties be $P_1, \ldots, P_n$ and let $\mathcal{I} \subset [n]$ denote the indices of corrupted parties, controlled by an adversary $\mathcal{A}$. We consider the real model in which a real n-party protocol π is executed (and there exist no trusted third party). In this case, the adversary $\mathcal{A}$ sends all messages in place of corrupted parties, and may follow an arbitrary polynomial-time strategy. The honest parties follow the instructions of π.

Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality where $f = (f_1, \ldots, f_n)$, and let π be an n-party protocol for computing f. Let $\mathcal{A}$ be a probabilistic polynomial time machine and $\mathcal{I} \subset [n]$ denote the indices of corrupted parties, controlled by an adversary $\mathcal{A}$. In addition, $\mathcal{A}$ gets access to a leakage oracle. Execution in the real model proceeds in two phases: Input Encoding phase, and Secure Computation phase. In the Encoding phase, each party P_i, locally computes an encoding of its input. The adversary has no access to the oracle during the encoding phase. In the Secure Computation phase, the parties begin to interact, and the adversary gets access to the leakage oracle. We abbreviate by $\mathcal{O}_i^\lambda$, the oracle $\mathcal{O}_{s_i}^\lambda$, where s_i is the circuit for local computation of the ith party. At the end of each round j of the protocol, the adversary can query the oracle $\mathcal{O}_i^\lambda$ with any function h_j, for every honest party P_i. The function h_j is applied on the jth round computation transcript of the honest party P_i. At the end of round j, the adversary gets to see the leakages $l_i^j = \{h_j(c_i^j)\}_{1 \le i \le n, i \notin \mathcal{I}}$, where c_i^j denotes the transcript of computation in round j of party P_i. The adversary can even query with different functions h_j for every party P_i, and adaptively decide the leakage functions based on already obtained leakages. Then the real execution of π on inputs $\bar{x}$, auxiliary input z to $\mathcal{A}$, and given oracle access to $\mathcal{O}_i^\lambda$,$(i \notin \mathcal{I})$ and security parameter κ, denoted $REAL_{\pi,\mathcal{A}(z),\mathcal{I}}^{\mathcal{O}_i^\lambda}(\bar{x}, \kappa)$, is defined as the output vector of the honest parties and the adversary $\mathcal{A}$ from the real execution of π.

We can even think of a stronger variant of the above model in which, the leakage function could be a joint function of the transcripts of all honest parties. When $\lambda = 1$, we call this the *complete* leakage oracle. We also observe that, in the complete leakage case, the two variants: Joint leakage function and individual leakage function models are equivalent.

B Building Blocks

We briefly define the cryptographic primitives that are used in our constructions.

Fully Homomorphic Encryption. We make use of a Fully Homomorphic Encryption scheme (FHE) in our protocol.

Definition 5. *The scheme $\mathcal{E} = (KeyGen, Encrypt, Decrypt, Eval)$ is correct for a circuit $\mathcal{C}$, if for any key pair (pk, sk) output by $KeyGen(\kappa)$, any plaintext bits $m_1, \ldots, m_k$ and any ciphertext $c = (c_1, \ldots, c_k)$, where $c_i \leftarrow Encrypt(m_i, pk)$, the following holds:*

$$Decrypt(Eval(c, \mathcal{C}, pk), sk) = \mathcal{C}(m_1, \ldots, m_k)$$

$\mathcal{E}$ is fully homomorphic if it is correct for all boolean circuits, and the ciphertext size and decryption time are upper-bounded independently of $\mathcal{C}$.

Witness Indistinguishable Arguments. An interactive proof for a language $L \in NP$ is *witness indistinguishable* if the verifier's view of the interaction with the prover is computationally independent of the auxiliary input of the prover [Gol01].

Definition 6. *Let (P,V) be an interactive proof for a language $L \in NP$, and let R_L be a witness relation for the language L. We say (P,V) is witness-indistinguishable for R_L, if for every probabilistic polynomial time interactive machine V, and every two sequences $W^1 = \{w_x^1\}_{x \in L}$ and $W^2 = \{w_x^2\}_{x \in L}$, such that $w_1^x, w_2^x \in R_L(x)$, the ensembles resulting when the auxiliary input of P is w_1^x, w_2^x, are computationally indistinguishable. The following conditions hold:*

- *Indistinguishability: For every PPT algorithm D, all $x \in L$, and all $z \in \{0,1\}^*$,*

$$|Pr[D(x, z, \langle P(w_x^1), V(z)\rangle(x)) = 1] - Pr[D(x, z, \langle P(w_x^2), V(z)\rangle(x)) = 1]| \leq \frac{1}{p(|x|)}$$

- *Completeness: For every $x \in L$, there exists a string y, such that for every z $Pr[\langle P(y), V(z)\rangle(x) = 1] \geq \frac{2}{3}$.*
- *Soundness: For every $x \notin L$, and for every y, z, the $Pr[\langle P(y), V(z)\rangle(x) = 1] \leq \frac{1}{3}$.*

In our construction, we use a witness indistinguishable argument (WI) for proving membership in any NP language with perfect completeness and negligible soundness error, δ.

Commitment scheme. A commitment scheme is an efficient two-phase protocol through which one party called the sender, commits itself to a value such that the following requirements are satisfied.

- Hiding: At the end of the first phase, the other party, receiver does not gain any knowledge of the sender's value.
- Binding: Given the transcript of the interaction in the first phase, there exists at most one value that the receiver can later accept as a legal opening of the commitment.

We use a non-interactive statistically binding commitment scheme, COM in our protocol. Such a scheme can be constructed using one-way permutations.

Yao's Garbled Circuit construction. Yao presented a constant round protocol for securely computing any functionality in the presence of semi-honest adversaries. The basic idea behind Yao's protocol is to provide a method for computing a circuit so that values obtained on all wires other than circuit-output wires are never revealed.

Garbled circuit construction: For every wire w in the circuit, two keys k_w^0 and k_w^1 are chosen, where k_w^σ represents the bit σ. The four possible inputs to a gate k_1^0,k_1^1,k_2^0,k_2^1 are viewed as encryption keys. The output values k_3^0 and k_3^1, which are also keys are encrypted under the appropriate keys from the incoming wires. A garbled circuit consists of garbled gates along with output decryption tables. These tables map the random values on circuit-output wires back to their corresponding real values. Given the keys associated with inputs wires, it is possible to obliviously compute the entire circuit gate-by-gate. We make use of the garbled circuit construction, and rely on its security [LP04] in our protocol.

C The Basic Construction: Missing Details

C.1 Indistinguishability Argument for Theorem 1

We prove the indistinguishability of the views going from the real protocol to the simulated one through a series of hybrids. We prove that the real protocol view is indistinguishable from that in the simulated execution by proving indistinguishability between every pair of successive hybrids.

- **Hybrid** H_0: This is the output distribution of a real execution of the protocol. Clearly, H_0 is identical to $REAL^{\mathcal{O}_2}_{C_\Pi,\mathcal{A}(z)}(\bar{x},\kappa)$.

- **Hybrid** H_1: This is identical to H_0 except in the leakages. The simulator replaces the leakage in the following way: A fake garbled circuit, $\widehat{GC}$ of the Next message function of the honest party in protocol Π_{INT} is constructed that always outputs the correct next message and the encrypted state as in the previous hybrid. The leakage is set to the evaluation transcript of this fake circuit. The leakage in all rounds is replaced by this simulated leakage.

  In distribution $H_{1,i}$, the first i leakages are evaluation of fake circuits, and the rest are real. We have the sub hybrids $H_{1,0},\ldots,H_{1,t}$ with $H_{1,0}=H_0$ and $H_{1,t}=H_1$.

  Indistinguishability from H_0, $H_0 \stackrel{c}{\equiv} H_1$: The two hybrids differ only in the leakage. H_0 consists of the evaluation of correct garbled circuits whereas, in H_1, fake garbled circuits are evaluated. For contradiction, assume there is a distinguisher D and a polynomial p, such that

  $$|Pr[D(H_{1,0})=1]-Pr[D(H_{1,t})=1]|>\frac{1}{p(\kappa)}$$

  it follows that, $\exists i$ such that,

  $$|Pr[D(H_{1,i})=1]-Pr[D(H_{1,i+1})=1]|>\frac{1}{tp(\kappa)}$$

  Two neighbouring hybrids $H_{1,i}$ and $H_{1,i+1}$ differ only in the $i+1$st transcript. The $i+1$st transcript is the evaluation of real Garbled circuit in $H_{1,i}$, whereas it is simulated in $H_{1,i+1}$. A distinguisher can be constructed that can distinguish between the distribution ensemble consisting of $\widehat{GC}$ and a single garbled value for each input wire, and the distribution ensemble consisting of a real garbled version of C, together with garbled values corresponding to the real input. This contradicts the security of Yao's garbled circuit protocol.

- **Hybrid** H_2: This is the same as H_1, except that in the leakages: the simulator changes the output of the fake garbled circuit to the encryption of a random string instead of the correct encrypted state. S internally maintains the correct state to honestly run Π, but only in the leakage, the fake garbled circuit outputs the encryption of a random string. In distribution $H_{2,i}$, the fake garbled circuits of the first i leakages output the encryption of a random string, and the rest are real. $H_{2,0}$ is the same as H_1, and hybrid $H_{2,t}$ is H_2.

  Indistinguishability from H_1, $H_1 \stackrel{c}{\equiv} H_2$: Assume the existence of a distinguisher D that can distinguish between $H_{2,0}$ and $H_{2,t}$ it follows that, $\exists i$ such that, D that can distinguish between $H_{2,i}$ and $H_{2,i+1}$ Two neighbouring hybrids $H_{2,i}$ and $H_{2,i+1}$ differ only in the $i+1$st transcript. The $i+1$st transcript is the evaluation of a fake garbled circuit that outputs the correct encrypted state in $H_{2,i}$ and encryption of a random string in $H_{2,i+1}$. D can be used to distinguish two ciphertexts under the encryption scheme E. This contradicts the semantic security of E.

- **Hybrid** H_3: In this experiment, S receives the commitments from A, sends a random challenge e, and gets the openings. If it is non accepting transcript (the openings are incorrect), S outputs $\perp$ and aborts. Otherwise, S keeps rewinding A with random challenge strings, until it gets another accepting transcript with another challenge e'. If $e=e'$, S outputs $\perp$, and aborts. Otherwise, by the responses of A to two distinct challenges

e, e', the appropriate shares are combined and the adversary's input and randomness X_1' is recovered. S now plays honest committer, and commits to X_2' and R_2' and runs rest of the protocol honestly.

Indistinguishability from H_2:In experiment H_3, everything is identical to H_2 except that S rewinds A and extracts the input and randomness. S aborts in H_3, whenever it fails to extract X_1'. The two hybrids are indistinguishable conditioned on the event that S does not abort in H_3. S fails and aborts if the challenge strings for the two accepting transcripts in the challenge response phase are the same, that is, Pr[S aborts] = Pr[$e = e'$] = $\frac{1}{2^k}$. Thus, S aborts in H_3 with negligible probability, and therefore, $H_3 \stackrel{s}{\equiv} H_2$.

- **Hybrid** H_4: In this experiment, S rewinds A, and extracts the input and randomness X_1' in Phase I as before. Now, S plays honest committer and commits to random shares of the honest party's input and randomness, X_2'. In the next step, instead of committing to shares of a random string R_2', S gives commitments to random shares of X_1'. S now runs the rest of the protocol honestly.

  Indistinguishability from H_3, $H_3 \stackrel{c}{\equiv} H_4$: Hybrid H_4 is the same as hybrid H_3 except in the commitment to random string R_2'. S gives commitments to random shares of R_2' in H_3, whereas, in H_4, it commits to random shares of the extracted input and randomness X_1'. If there exists a PPT distinguisher that can distinguish H_3 and H_4, then there is a distinguisher D that can distinguish between commitments to two strings α and α' under the commitment scheme COM. This contradicts the hiding property of the scheme COM.

- **Hybrid** H_5: In this experiment, phase I is run as in the previous hybrid. In phase II, S runs protocol Π honestly and in all WI arguments it proves the statement:

EITHER (a) (i) there exists values $X_2' = (x_2', r_2')$, such that the shares committed to phase 1 can be decommitted to shares of X_2', and (ii) the sent message M_2^j is consistent with input x_2' and randomness r_2'', that is, $M_2^j = \Pi(T_2^j, x_2', r_2'')$,

OR (b) The shares of the random string R_2' it committed to in phase 1, are shares of X_1'.

  In the proof, S uses the witness corresponding to the *second part* of the statement in all WI arguments associated with the messages of Π. This witness is available to S as it has already rewound A in phase I, and extracted X_1'.

  In distribution $H_{5,i}$, the first i WIs in the protocol, use the second part of the statement as witness, and the rest use the witness corresponding to the first part. Hybrid $H_{5,0}$ is the same as H_4, and $H_{5,t}$ is the same as H_5.

  Indistinguishability from H_4, $H_4 \stackrel{c}{\equiv} H_5$: For the sake of contradiction, assume the existence of a distinguisher D that can distinguish between H_5 and H_4. This implies, $\exists i$, such that D can distinguish between $H_{5,i}$ and $H_{5,i+1}$. D can be used to distinguish between two WI arguments using two different witnesses w and w' with the same probability with which $H_{5,i}$ and $H_{5,i+1}$ are distinguished. This contradicts the witness indistinguishability property of WI arguments. Therefore, H_4 and H_5 are distinguished with probability $\leq r.\gamma$, where r is the number of rounds in protocol Π, and γ is the probability that WI arguments with two different witnesses are distinguishable, which is negligible.

- **Hybrid** H_6: In this experiment, during the commit phase, S gives commitments to shares of random strings in phase I instead of committing to shares of correct input and randomness of the honest party. Rest is identical to the previous hybrid.

  Indistinguishability from H_5, $H_5 \stackrel{c}{\equiv} H_6$: The hybrids H_6 and H_5 differ only in the commitments to X_2' in phase I. The commitments to random strings $X_2' = (x_2', r_2')$ in H_5 are changed to other random strings in H_6. These commitments are not used elsewhere in the experiment (S uses the fake witness in the WIs).

- **Hybrid** H_7: In this experiment, S simulates the execution of Π. Let S_Π be the simulator for the semi honest protocol Π. S runs S_Π with the extracted input instead of running the protocol Π honestly, and gets the transcript T and an associated randomness. S now forces the transcript T and randomness on the adversary

as described in section 4.1. If A does not reply according to T and succeeds in the associated WI, S aborts. The output of the fake garbled circuits in the leakage are also changed from the correct next message to the ones output by S_Π.

Indistinguishability from H_6, $H_6 \stackrel{c}{\equiv} H_7$: The hybrids H_7 and H_6 differ in phase II of the protocol. In H_6, the protocol Π is run honestly whereas in H_7, the transcript output by the simulator for Π, S_Π is used. If there is is a PPT distinguisher that can distinguish between H_7 and H_6, then the distinguisher D can be used to distinguish between a real execution of protocol Π and a simulated one, which contradicts the security of Π. Indistinguishability therefore follows from the security of Π and the negligible probability of S aborting in experiment H_7. Probability of A sending a message, which is not according to the transcript T but succeeding in WI is equal to the soundness error of WI which is negligible. This is because A will not have the witness to part (b) of the proof statement. If A is able to successfully commit to the input (and randomness) of the honest party and then succeed in the challenge response phase, it is easy to construct an extractor to extract such input (and randomness) from A and violate the hiding property of the commitment scheme COM.

Therefore, the probability of S aborting in experiment H_7 is $\leq r.\delta$, where r is the number of rounds in protocol Π, and δ is the soundness error of WI argument.

Therefore $H_7 \stackrel{c}{\equiv} H_0$. The hybrid H_7 is the same as the execution of the simulator S, and we have seen that H_0 is the real execution of C_Π. It follows that C_Π is secure as per Definition 4.

$$IDEAL_{f,S(z)}(\bar{x}, \kappa) \stackrel{c}{\equiv} REAL^{\mathcal{O}_2}_{C_\Pi, \mathcal{A}(z)}(\bar{x}, \kappa)$$

C.2 Extension to Multi-party case.

We now sketch the extension of protocol C_Π to the multi-party case. Let the n parties be denoted by $\{P_1, \ldots, P_n\}$, each party P_i holds input x_i. Given a semi-honest multi-party protocol Π, it is first compiled into an intermediate protocol Π_{INT} in the following way:
Phase I. For each party P_i:

- Let $X_i = (x_i, r_i)$ where r_i is the randomness of P_i. Party P_i breaks the string X_i into k pairs of random shares, and gives commitments to these shares to all the other parties.
- The rest of the parties run a multi-party coin tossing protocol to jointly generate a uniformly random string e_i. Party P_i opens the commitments to one share in each pair of commitments as determined by the challenge string.
- Party P_i picks a random string R_i, and similarly commits to secret shares of R_i, gets a challenge string generated jointly by all parties and opens commitments corresponding to the challenge string.

Phase II. Now, each message to be sent by P_i in Π is compiled into a message block in Π_{INT}.

- P_i sends the next message m_i^j as per protocol Π.
- P_i now engages in a WI with every other party P_j (sequentially), and proves the statement: either
  (a) there exists $X_i = (x_i, r_i)$ such that commitment in step (1) can be decommitted to X_i, and, the sent message is consistent with the input and randomness, or
  (b) The shares of random string R_i are the shares of X_j.

This protocol Π_{INT} is compiled with the compiler C to get a leakage resilient multi-party protocol C_Π. That is, each of the parties constructs garbled circuits corresponding to the next message function of the protocol Π_{INT} in the input encoding phase and uses them to compute the next message in the secure computation phase as described before.

D Proof of Theorem 2

W.l.o.g., let P_1 be the corrupt party in C_Π. Let A be the adversary controlling P_1 in C_Π. The simulator S works in a similar way as described in Section 4.1, except for the way leakage is simulated.

1. S chooses a key pair of the fully homomorphic encryption scheme E.
   $(pk_s, sk_s) \leftarrow KeyGen(1^\kappa)$
2. Let m_1^{j-1} be the message sent by A in round $j-1$. Then, S encrypts m_1^{j-1}, and sets l_e to be the transcript of encryption.
3. Next, S picks a random string $rand_j$ and computes $Eval(E[rand_j], E[m_1^{j-1}], NextMsg_2^j)$. It sets l_h to be the transcript of this homomorphic evaluation of $NextMsg_2^j$.
4. S also constructs a fake garbled circuit $\widehat{GC}$ of the decryption circuit that always outputs m_2^j and sets l_d to be the transcript of evaluation of this garbled circuit. (m_2^j is the message S sends to the adversary in round j, simulated as described in Section 4.1).
5. The leakage l_2^j is set to be $\langle l_e, l_h, l_d \rangle$. This leakage l_2^j is sent to A along with message m_2^j.
6. Whenever S rewinds the adversary and gives leakage for round j again, S constructs the entire leakage again from scratch using a fresh random tape (including constructing a fresh fake garbled circuit of the decryption circuit).

We prove the indistinguishability of the views going from the real protocol to the simulated one through a series of hybrids. We prove that the real protocol is indistinguishable from the simulated execution by proving indistinguishability between every pair of successive hybrids.

- **Hybrid** H_0: This hybrid corresponds to the real execution.
- **Hybrid** H_1: This hybrid is identical to the previous hybrid except for the l_d component of leakage. The simulator constructs a fake garbled circuit of the decryption circuit, $\widehat{GC}$, that always outputs m_2^j. It evaluates these garbled circuits under the wire keys corresponding to $E[m_2^j]$ and sets l_d of each leakage to the transcript.

  Indistinguishability from H_0: $H_0 \stackrel{c}{\equiv} H_1$

  Let $H_{1,i}$, $0 \leq i \leq t$, be the distribution in which the l_d components of the first i leakages are simulated, and the rest are real. Note that $H_{1,0} = H_0$ and $H_{1,t} = H_1$. Hence, $H_{1,i}$ and $H_{1,i+1}$ differ only in the $i+1$st transcript. The l_d component of the $i+1$st leakage is the transcript of the evaluation of a real garbled circuit in $H_{1,i}$ whereas, it is a fake circuit in $H_{1,i+1}$. By standard arguments (as we have seen before), a distinguisher for H_0 vs. H_1 can also distinguish (with at most a factor of t loss in the distinguishing probability) between a real garbled circuit and a fake one. This contradicts the security of the garbled circuit protocol.
- **Hybrid** H_2: This hybrid is the same as the previous one, except for the following: The simulator picks a random string $rand_j$, and evaluates the $NextMsg_2^j(E[m_1^j], E[rand_j])$ function homomorphically (using the scheme chosen by the simulator). The l_h component of each leakage l_2^j is replaced by the transcript of the above evaluation by the simulator. The l_e component remains unchanged. The simulator maintains the correct state internally, to run the rest of the protocol honestly.

  Indistinguishability from H_1: $H_1 \stackrel{c}{\equiv} H_2$

  H_2 differs from H_1 only in the l_h leakage components. The homomorphic evaluation of the Next Message function is done honestly in H_1, whereas in H_2, this evaluation uses the encryption of a random string instead of the correct encrypted state. The indistinguishability from H_1 follows from the indistinguishability of ciphertexts under the encryption scheme E.
- Rest of the hybrids are the same as the hybrids H_3 – H_7 in the previous protocol.

It follows that $H_7 \stackrel{c}{\equiv} H_0$. The hybrid H_7 is the same as the execution of the simulator S. Using S as an ideal world adversary for f and recalling that H_0 corresponds to the real execution of C_Π with adversary A, we conclude that the ideal world execution of f and the real world execution of C_Π are computationally indistinguishable. This proves Theorem 2. ■

E Continual Leakage Resilient Protocol: The Model

We give a formal definition of security against malicious adversaries in the presence of continual leakage in the ideal/real simulation paradigm. The execution in the ideal and real world are as follows:

Execution in the Ideal Model

Inputs: Each party obtains an input; the i^{th} party's input is denoted by x_i; we assume that the inputs are of the same length n. The adversary $\mathcal{A}$ receives an auxiliary-input z.
Send inputs to trusted party: The honest party P_j sends its received input x_j to the trusted party. A malicious party may depending on its input (as well as on its coin tosses and the auxiliary input) either abort or send some other input of the same length to the trusted party.
Trusted party answers the first party: In case the trusted party received the input pair, (u_1, u_2), the trusted party first replies to the first party with $f(u_1, u_2)$.
Trusted party answers the second party: In case the first party is malicious, it may depending on its input and the trusted party's answer, decide to abort the trusted party. In this case, the trusted party sends abort to the second party. Otherwise, the trusted party sends $f(u_1, u_2)$ to the second party.
Outputs: An honest party always outputs the message it obtained from the trusted party. A malicious party may output any arbitrary (probabilistic polynomial-time computable) function of the initial input and the message obtained from the trusted party.
k **interactions:** The parties interact k times, and compute a different functionality of the same inputs x_i. That is $f_j(x_1, x_2)$ is the functionality computed in the jth interaction with the trusted party.

Definition 7. *The outputs of honest party and the adversary in k executions of the above model is denoted by $IDEAL_{k,f,\mathcal{A}(z)}(x_1, x_2)$.*

Execution in the Real Model

Let P_1 and P_2 be two parties who wish to securely compute functionality $f : (\{0,1\}^*)^2 \to (\{0,1\}^*)^2$. W.l.o.g., let P_1 be the corrupt party (P_2 is honest). We want to model leakage from the honest party P_2. Note that we only need to consider leakage from honest parties. Let $\mathcal{A}$ be a PPT adversary controlling malicious party P_1. The adversary $\mathcal{A}$ can deviate from the protocol in an arbitrary way. In particular, $\mathcal{A}$ can access leakage from P_2 via its leakage oracle throughout the execution of the protocol. In addition, $\mathcal{A}$'s behavior may depend on some auxiliary input z. More details follow.

Our real world execution proceeds in three phases: a leakage-free *Encoding phase* and a *Secure Computation Phase* and a *Refresh Phase* in which leakage may occur. In the Encoding phase, each party P_i, locally computes an encoding of its input and any other auxiliary information. The adversary has no access to the leakage oracle during the encoding phase. In the Secure Computation phase, the parties begin to interact, and the adversary gets access to the leakage from the local computation of P_2. Specifically, let s_2 be a circuit describing the internal computation of P_2. The circuit s_2 takes as input the history of protocol execution of the *secure computation* phase so far, and some specified bits of the encoded input, and outputs the next message from P_2 and the new internal state of P_2. Thus, by Definition 1, the leakage is accessible to the adversary via the oracle $\mathcal{O}_{s_2}$, which we abbreviate by $\mathcal{O}_2$. Thus, at the end of round j of the protocol, $\mathcal{A}$ can query $\mathcal{O}_2$ for any PPT function h_j applied on the transcript of the jth round computation of the honest party P_2. We denote this leakage information by l_2^j. After each execution, the parties run a *Refresh phase*. The adversary continues to have access to the leakage oracle during the Refresh phase. After Refresh, the protocol, that is the Secure Computation phase can be run again on the same input. Specifically, the *Encoding phase* is run just once in the beginning for k executions of the protocol. The *Refresh phase* is run after each execution of the *Secure Computation phase*.

Definition 8. *We define* $REAL^{\mathcal{O}^\lambda}_{k,\pi,\mathcal{A}(z)}(x_1, x_2, \kappa)$ *to be the output pairs of the honest party and the adversary* $\mathcal{A}$ *from* k *real executions of* π *as defined above on inputs* (x_1, x_2)*, auxiliary input* z *to* $\mathcal{A}$*, with oracle access to* $\mathcal{O}_2^\lambda$*, and security parameter* κ*.*

Definition of Security

Definition 9. *Let* f, π*, be the executions in the ideal world and the real world as described above. Protocol* π *is said to* securely compute f in the presence of $\lambda-$ continual leakage *if for every non-uniform probabilistic polynomial-time pair of algorithms* $\mathcal{A} = (A_1, A_2)$ *for the real model, there exists a non-uniform probabilistic polynomial-time pair* $S = (S_1, S_2)$ *for the ideal model such that*

$$IDEAL_{k,f,S(z)}(x_1, x_2) \stackrel{c}{\equiv} REAL^{\mathcal{O}^\lambda}_{k,\pi,\mathcal{A}(z)}(x_1, x_2, \kappa)$$

for a polynomial number of executions k*, where* $x_1, x_2, z \in \{0,1\}^*$*, such that* $|x_1| = |x_2|$ *and* $|z| = \mathsf{poly}(|x_1|)$*. When* $\lambda = 1$ *in the Secure Computation phase, protocol* π *is said to* securely compute f in the presence of complete leakage.