On Concurrently Secure Computation in the Multiple Ideal Query Model*

Vipul Goyal† Abhishek Jain‡

Abstract

The multiple ideal query (MIQ) model was introduced by Goyal, Jain and Ostrovsky [Crypto'10] as a relaxed notion of security which allows one to construct concurrently secure protocols in the plain model. The main question relevant to the MIQ model is how many queries must we allow to the ideal world adversary? The importance of the above question stems from the fact that if the answer is positive, then it would enable meaningful security guarantees in many application scenarios, as well as, lead to resolution of long standing open questions such as fully concurrent password based key exchange in the plain model.

In this work, we continue the study of the MIQ model and prove severe lower bounds on the number of ideal queries per session. Following are our main results:

1. There exists a two-party functionality that cannot be securely realized in the MIQ model with only a constant number of ideal queries per session.
2. There exists a two-party functionality that cannot be securely realized in the MIQ model by any constant round protocol, with *any polynomial number of ideal queries* per session.

Both of these results are unconditional and even rule out protocols proven secure using a non-black-box simulator. We in fact prove a more general theorem which allows for trade-off between round complexity and the number of ideal queries per session. We obtain our negative results in the following two steps:

1. We first prove our results with respect to *black-box* simulation, i.e., we only rule out simulators that make black-box use of the adversary.
2. Next, we give a technique to "compile" our negative results w.r.t. black-box simulation into full impossibility results (ruling out *non-black-box* simulation as well) in the MIQ model. Interestingly, our compiler uses ideas from the work on obfuscation using tamper-proof hardware [GIS+10, GO96], even though our setting does not involve any hardware tokens.

*This is the full version of the Eurocrypt'13 paper.

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

‡MIT and Boston University. Email: abhishek@csail.mit.edu. This material is based on research sponsored by NSF grant #1218461 and DARPA under agreement number FA8750-11-2-0225. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government. The author also thanks RISCS (Reliable Information Systems and Cyber Security) institute.

1 Introduction

The notion of secure computation is central to cryptography. Introduced in the seminal works of [Yao86, GMW87], secure multi-party computation allows a group of (mutually) distrustful parties $P_1, \ldots, P_n$, with private inputs $x_1, \ldots, x_n$, to jointly compute any functionality f in such a manner that the honest parties obtain correct outputs and no group of malicious parties learn anything beyond their inputs and prescribed outputs.

The classical results for secure computation are only in the *stand-alone* setting where security holds only if a single protocol session is executed in isolation. Unfortunately, as it has become increasingly evident over the last two decades, stand-alone security does not suffice in real-world scenarios where several protocol sessions may be executed *concurrently* – a typical example being protocols executed over modern networked environments such as the Internet.

Background: Concurrently Secure Computation. Towards that end, the last decade has a seen a significant effort by the cryptographic community towards obtaining protocols that are *concurrently composable*, i.e., protocols that remain secure even when executed concurrently over an insecure network. For example, we could require security under *concurrent self-composition* (which is the focus of this work): a protocol should remain secure even when there are multiple copies executing concurrently. The framework of universal composability (UC) [Can01] was introduced to capture the setting of *concurrent general composition*, where a protocol may be executed concurrently not only with several copies of itself but also with other arbitrary protocols.

General positive results for UC secure computation are known based on various trusted setup assumptions, such as a common random string [Can01, CF01, CLOS02, BCNP04, CPS07, Kat07, LPV09]. Whether a given set of players is actually willing to trust an external entity, however, is debatable. Indeed, a driving goal in cryptographic research is to eliminate the need to trust other entities. As such, positive results for concurrently-secure computation in the *plain model* (which is the main focus of this work) are highly desirable, both from a theoretical and practical viewpoint.

Negative Results for Concurrent Composition. Unfortunately, in the plain model, by and large, most of the results have been negative. UC secure protocols for most functionalities of interest were ruled out in [CF01, CKL03, PR08, KL11]. These impossibility results were extended to the setting of general composition by Lindell [Lin03] who proved that security under concurrent general composition implies UC security. Later, Lindell [Lin04] established broad negative results even for the setting of concurrent self-composition by showing equivalence of concurrent self-composition and general composition for functionalities where each party can "communicate" to the other via its output. Following the work of Barak et al. [BPS06] and Goyal [Goy12], recently Agrawal et al. [AGJ+12] and Garg et al. [GKOV12] ruled out essentially all non-trivial two-party functionalities for concurrent self-composition, even in the setting where the inputs of honest parties are fixed in advance for all the protocol sessions.

On the positive side, it is known how to realize zero-knowledge and related functionalities, with security in similar models (e.g., [DNS98, RK99, KP01, PRS02, BPS06, Lin08]). The recent work of Goyal [Goy12] obtains positive results for a broader class of functionalities; however, (in keeping with the negative results mentioned above) these results are only relevant to the restricted setting where an honest party uses the *same*, static input in all of the sessions.

The Search for Relaxed Security Notions. While the above discussion paints a rather bleak picture of state of the art on concurrent security, fortunately, there is a brighter side. Indeed, several prior works have studied relaxations of the standard definition of secure computation that bypass the above negative results, yet provide strong and meaningful security guarantees in the concurrent setting. A well studied notion is that of security w.r.t. super-polynomial simulation [Pas03, PS04, BS05, LPV09, CLP10, GGJS12, LP12] which intuitively guarantees that a real-world adversary does not learn any more information than what can be computed in super-polynomial time in the ideal world. Another notion is that of input-indistinguishable computation [MPR06, GGJS12] which intuitively guarantees that an adversary cannot decide which input (out of possibly many inputs leading to the same output) is used by the honest party in the protocol.

Recently, Goyal, Jain and Ostrovsky [GJO10] introduced the *multiple ideal query* (MIQ) model for concurrent self-composition where the ideal world adversary is allowed to make more than one output query per session to the ideal functionality. The exact number of queries allowed is a priori fixed by a parameter λ. In our view, the main advantage of this notion over the previously discussed notions is that it provides an (arguably) intuitive, easy to understand, security guarantee. In particular, in this model, one can precisely measure the amount of "extra" information that the adversary can potentially learn. The security guarantee is provided with standard polynomial time simulation (and adversary) and follows the ideal/real world security formalization. Furthermore, for functionalities such as password-based key exchange, the MIQ definition in fact implies the previous standard definition [GL01] when $\lambda = \mathsf{O}(1)$.

The MIQ model has also proven relevant in the related setting of resettability. Goyal and Sahai [GS09] introduced the notion of resettable secure computation and construct such protocols in the model where the ideal adversary can "reset" the trusted party at any point. This gives the ideal adversary the power to query the trusted party multiple times (per session in the real world). This allows them to get a general positive result for all PPT computable functionalities in the plain model in the resettable ideal world setting.

The multiple ideal query model has also proven relevant as technical tool. In particular, the recent positive results of Goyal [Goy12] can be seen as obtained using the following two step paradigm. First, very roughly, Goyal constructs a protocol secure in the multiple ideal query model. Then, the additional queries made to the ideal trusted party are eliminated by constructing an "output predictor".

We believe the study of the MIQ model is well motivated: both because the guarantee provided in the concurrent setting is interesting and non-trivial on its own, as well as the connection it has to constructing secure protocols in other related settings.

In this work, we continue the study of the MIQ model.

Our Question: How many Queries? The main question relevant to the MIQ model is how many queries must we allow to the ideal world adversary? Note that if we allow a large number of queries, then the security guarantee may quickly degrade and become meaningless; in particular, the adversary may be able to learn the input of the honest party in the worse case. On the other hand, if the number of allowed queries is very small, say $1+\epsilon$ per session, then the security guarantee is very close to that of the standard definition.

To exemplify this further, consider the oblivious polynomial evaluation functionality [NP99, NP06] where two parties wish to jointly evaluate a polynomial over a point. The input of party P_1 is a polynomial Q, while the input of P_2 is a point α. At the end of the protocol, the party P_2 gets

$Q(\alpha)$ as the output. This is a natural functionality with applications to list intersection, mutual authentication, metering on the web, etc (see [NP06] for more details on these).

Now, note that if we only allow, say, 2 queries to a malicious P_2 in the ideal world (per real world session), then as long as Q is a high-degree polynomial, the security guarantee for P_1 is still quite meaningful. Instead of a single point, now a malicious adversary may learn the output on two points of its choice (from an exponential domain of points). The adversary still does not learn any information about what the polynomial evaluates to on rest of the (exponential) domain. On the other hand, if we allow too many queries (exceeding the degree of the polynomial), then the ideal world adversary may be able to learn the entire polynomial Q thus rendering the security guarantee meaningless.

The only known positive results in the MIQ model are due to [GJO10, GGJ13]. Goyal et. al. [GJO10] provide a construction where the *average* number of queries in the ideal world per real world session is a constant (with the constant depending upon the adversary). This was further improved in a recent result [GGJ13] which provides a construction where the *average* number of ideal queries in any session are $(1 + \frac{1}{\text{poly}(n)})$.

If the guarantee on the number of queries per session is only in expectation, this means that in some sessions, the ideal adversary may still be able to make a large number of queries (while keeping the number of queries low in other sessions). Considering the oblivious polynomial evaluation example above, this means that the security in some sessions may be *completely* compromised. Furthermore, consider the problem of concurrent password based key exchange [KOY01, GL03, CHK$^+$05, BCL$^+$05, GJO10]. An interesting question that has remained open so far is designing a concurrent password based key exchange in the plain model where different pair of parties share different (but correlated) passwords. Indeed, this is the most natural setting for password based key exchange (PAKE) and all currently known construction either do not provide concurrent security or are not in the plain model. We note that a positive result in the MIQ model where the number of queries per sessions is a *strict constant* would directly imply a positive result for this problem.[1] This raises the following natural question:

"*Do there exist concurrent secure protocols in the MIQ model where the number of ideal queries per session is a strict constant?*"

As discussed, the importance of the above question stems from the fact that if the answer is positive, then it would enable meaningful security guarantees in many application scenarios (e.g., oblivious polynomial evaluation), as well as, lead to resolution of long standing open questions such as concurrent password based key exchange in the plain model.

1.1 Our Results

In this work, we continue the study of the MIQ model and prove severe unconditional lower bounds on the number of ideal queries per session. Following are our main results:

1. There exists a two-party functionality that cannot be securely realized in the MIQ model with only a constant number of ideal queries per session.

[1] The first positive result for concurrent PAKE in the plain model provided by Goyal et. al. [GJO10] was for the *single* password setting where there is a single global correct password which every party is required to use for authentication. The single password restriction stems from their solution requiring a constant number of ideal queries per session on an *average*.

2. There exists a two-party functionality that cannot be securely realized in the MIQ model by any constant round protocol, with *any polynomial number of ideal queries* per session.

Both of these results are unconditional and even rule out protocols proven secure using a non-black-box simulator. We in fact prove a more general theorem that provides a trade-off between the round-complexity and the number of ideal queries per session.

Let $\mathsf{ceil}_1(x) = \lceil x \rceil$. For $y \geq 2$, let $\mathsf{ceil}_y(x)$ be recursively defined as $\mathsf{ceil}_y(x) = \lceil x \cdot \mathsf{ceil}_{y-1}(x) \rceil$. Our main result is stated as follows:

Theorem 1 (Informal) *There exists a two-party functionality f such that for any $d = d(k)$ and $n = n(k)$ that satisfy $n^d = \mathrm{poly}(k)$, no n-round protocol Π securely realizes f in the MIQ model with at most $\lambda = \mathsf{ceil}_d(1 + \frac{1}{n})$ number of queries per session.*

Application to concurrent precise zero-knowledge. While our main results concern with the MIQ model, interestingly, they also find applications in the setting of *precise simulation* [MP06]. Recall that in the setting of precise simulation, we wish to ensure that the resource utilization of the simulator is "close" to the resource utilization of the adversary in the real world interaction. The resource being studied is typically the running time, however, previous works have also considered a more general setting where the resource in question can be, e.g., memory. One can consider a general setting, where instead of focusing only on a particular resource (such as time), we consider many resources at the same time, such as memory, cache, power, etc. A general question we may ask is whether it is possible to perform simulation that achieves precision for each of these resources simultaneously.

To be more concrete, say we have a concurrent adversary interacting with the prover in many sessions and making use of k different resources (each resource may be utilized by the adversary at any arbitrary point). A natural question is: can one obtain a zero-knowledge simulator such that its utilization of *each* resource is only within a constant factor of the adversary? Our negative results directly imply a negative answer for this question as well where the number of resources is equal to the number of sessions. In other words, viewing the ideal functionality query in session i as a utilization of the i-th resource, we directly have that the simulator will end up going over a constant factor for at least one of the resources. Similarly, our results also imply a severe lower bound for constant round protocols: there exists a resource whose utilization will, in fact, not be within any polynomial factor of the adversary's utilization.

Independent of our work, Pass [Pas12] has been able to obtain a *positive* result in the *stand-alone* setting for the above problem. In particular, [Pas12] gives a construction where the simulator is able to be precise in multiple resources simultaneously in the standalone setting.

1.2 Our Techniques

In this section, we give an overview of our techniques. We obtain our negative results in the following two steps:

1. We first prove our results with respect to *black-box* simulation, i.e., we only rule out simulators that make black-box use of the adversary.

2. Next, we give a technique to "compile" our negative results w.r.t. black-box simulation into full impossibility results (ruling out *non-black-box* simulation as well) in the MIQ model.

Interestingly, our compiler uses ideas from the work on obfuscation using tamper-proof hardware [GIS+10, GO96], even though our setting does not involve any hardware tokens. We believe these techniques may find applications elsewhere.

Below, we discuss each of these steps separately.

Impossibility for Black-box Simulation. Recall that in order to prove security of a two-party computation protocol, we need to demonstrate that for every real world adversary $\mathcal{A}$ that controls one of the parties, there exists an ideal world adversary (or simulator $\mathcal{S}$) who can simulate the view of $\mathcal{A}$. Typically, the simulator $\mathcal{S}$ works by extracting the input a used by $\mathcal{A}$ and then querying the ideal functionality with a to receive the correct output; this output is then used to complete the simulation of $\mathcal{A}$'s view.[2] Now, further recall that the only advantage that a black-box simulator has over the real adversary is the ability to *rewind*. In other words, a black-box simulator extracts the input of $\mathcal{A}$ by rewinding. However, in the concurrent setting, extracting the input of $\mathcal{A}$ in each session is a non-trivial task. In particular, given an adversarial scheduling, it may happen that in order to extract the input of $\mathcal{A}$ in a given session s, the simulator $\mathcal{S}$ rewinds past the beginning of another session s' (that is interleaved inside the protocol messages of session s). When this happens, $\mathcal{A}$ may change its input in session s'. Thus, the simulator $\mathcal{S}$ would be forced to query the ideal functionality more than once for the session s'. Indeed, as shown in [Lin04], this intuition can be formalized to obtain a black-box impossibility result for concurrent self-composition w.r.t. the standard definition of secure computation, where only one query per session is allowed.

We now briefly explain how the above intuition can be further extended to achieve a black-box impossibility result even when the simulator is allowed to make multiple queries to the ideal functionality. For concreteness, let us consider the (simplified) case where the simulator is allowed a fixed *constant* C number of queries per session. Note that in order to obtain the desired negative result, we need to construct a concurrent adversary $\mathcal{A}$ that can force any black-box simulator $\mathcal{S}$ to make more than C queries for at least one session. We now briefly discuss how to construct such an adversary. Let n be the round complexity of the protocol, where n is any polynomial in the security parameter.

Consider the following *static* adversarial scheduling of messages. Consider an "outer" session (say) s. We will call it a session at level 0. Now, between every round of messages in the outer session, place a new complete protocol session. We will call these n sessions to be at level 1. Next, we again place a new complete protocol session between every round of messages in each session at level 1. Note that this creates n^2 sessions at level 2. Repeat this process recursively until we reach level C, where there are exactly n^C sessions. Thus, in total, we have $m = \frac{n^{C+1}-1}{n-1}$ sessions, which is polynomial in the security parameter.

Now, as discussed earlier, a black-box simulator $\mathcal{S}$ must perform at least one rewinding in order to extract the input of $\mathcal{A}$ in the "outer" session s. Suppose that $\mathcal{S}$ rewinds the i^{th} round of session s. Then, this immediately implies that the i^{th} session at level 1 is executed at least twice, which in turn means that $\mathcal{S}$ will be forced to query the ideal functionality twice for that session. Now, note that $\mathcal{S}$ will need to extract $\mathcal{A}$'s input in each of these two executions in order to complete them successfully. Thus, assuming that (even if) $\mathcal{S}$ rewinds different rounds in each of these two executions, we have that there exist two sessions at level 2 that are each executed three times.

[2]Note that since the output depends on the honest party's input, for many natural functionalities, it is easy to see that $\mathcal{S}$ *must* query the ideal functionality on $\mathcal{A}$'s input in order to obtain a "valid" output. Indeed, the validity of the output may be publicly verifiable, or in particular, may be verifiable given $\mathcal{A}$'s auxiliary input.

Continuing this argument inductively, we can show that for every level i, there exists at least one session that is executed $i+1$ times. As a result, we obtain that there exists a session s^* at level C that is executed $C+1$ times. If the adversary chooses a different input in each of the $C+1$ executions, we have that $\mathcal{S}$ must query the ideal functionality $C+1$ times for session s^*. Thus, we conclude that the black-box simulator $\mathcal{S}$, who is only allowed C queries, must fail.[3]

From Black-Box to Non-Black-Box. We now discuss a compilation technique to transform our negative results for black-box simulation into full impossibility results that rule out non-black-box simulation as well.

Recall that the main advantage that a non-black-box simulator has over a black-box simulator is that the former can make use of the adversary's code. Then, our high-level approach is to "nullify" this advantage by making use of secure program obfuscation. We note, however, that general program obfuscation is known to be impossible [BGI+01]. Towards this end, our key idea is to use positive results on program obfuscation using stateless tamper-proof hardware tokens by Goyal et. al. [GIS+10]. In the obfuscation with hardware model, one can take the given program and convert it into an obfuscated program using an obfuscation key k. The obfuscated program, for execution, would require oracle access to a hardware token having the obfuscation key k. We denote the functionality of the token (required to run the obfuscated program) by f_{token} (parameterized by the key k).

Very roughly, our idea is to implement the "token functionality" f_{token} of [GIS+10] using two-party computation. An important point is that f_{token} is "robust" to any polynomial number of queries; thus, it is particularly suited to the MIQ model. In more detail, we obtain our negative result in the following three steps described at a very high level (see Section 4 for details and further intuition):

Toy Experiment: Let Π be any protocol for the f_{token} functionality and let $\mathcal{A}$ be any concurrent adversary for Π that rules out black-box simulators that make at most λ queries per session. We first consider a toy experiment involving three parties, namely, Alice, Bob and David. In this experiment, Alice and Bob interact in multiple ideal world executions of the token functionality f_{token}. At the same time, Bob and David are involved in concurrent real-world executions of Π where Bob and David follow the same scheduling of messages as defined by adversary $\mathcal{A}$. Furthermore, the adversary Bob is allowed to *reset* David at any point during their interaction.

David, who has a secret input secret is instructed to reveal secret to Bob if all the executions of Π are completed "successfully". The goal of Bob is to successfully complete its interaction with David and learn the value secret. Then, the main idea in this experiment is that, by relying on our black-box impossibility result, we show that no adversarial Bob can succeed in learning secret, except with negligible probability.

Ideal World: In the second step, we eliminate David from the above experiment by obfuscating his next-message function in the f_{token}-hybrid model and give it as an auxiliary input to Bob (while the corresponding obfuscation "key" is given to Alice). This results in a scenario where Alice and Bob are the only parties, who interact in multiple ideal world executions of f_{token}. From the security of obfuscation, we can argue that this experiment can be reduced to the toy experiment; as such, no adversarial Bob can learn secret, except with negligible probability.

[3] We remark that in order to prove our general result, a more tight analysis is necessary; see the technical sections.

Real World Experiment: We finally consider the real world experiment, which is the same as previous step, except that all the ideal world invocations of f_{token} are now replaced with real-world executions of protocol Π. It is not difficult to see that in this experiment, an adversary Bob can simply play a "man-in-the-middle" between Alice and David (since Bob has David's obfuscated code); as a result, Bob can learn secret with probability 1.

From above, it follows that the adversary Bob in the real world experiment is a PPT concurrent adversary for Π whose view cannot be simulated by *any* simulator that makes at most λ queries per session, thus yielding us the desired result. We refer the reader to the technical sections for more details.

2 Preliminaries

2.1 Our Model

In this section, we present our security model. Throughout this paper, we denote the security parameter by k.

Concurrently Secure Computation in the MIQ model. We define our security model by extending the standard real/ideal paradigm for secure computation. Roughly speaking, we consider a relaxed notion of concurrently secure computation where the ideal world adversary is allowed to make an a priori fixed number (denoted as $\lambda = \lambda(k)$, where k is the security parameter) of output queries to the ideal functionality for each session. Note that in contrast, the standard definition for concurrently secure computation only allows for one output query per session to the ideal adversary. We now give more details.

In this work, we consider a malicious, static adversary. The scheduling of the messages across the concurrent executions is controlled by the adversary. We do not require fairness and hence in the ideal model, we allow a corrupt party to receive its output in a session and then optionally block the output from being delivered to the honest party, in that session. We consider a static adversary that chooses whom to corrupt before execution of the protocol. Finally, we consider *computational* security only and therefore restrict our attention to adversaries running in probabilistic polynomial time. We denote computational indistinguishability by $\stackrel{c}{\equiv}$.

We now proceed to describe the ideal and real world experiments and then give our security definition.

IDEAL MODEL. We first define the ideal world experiment, where there is a trusted party for computing the desired two-party functionality f. Let there be two parties P_1 and P_2 that are involved in multiple sessions, say $m = m(k)$. An adversary may corrupt either of the two parties. As in the standard ideal world experiment for concurrently secure computation, the parties send their inputs to the trusted party and receive the output of f evaluated on their inputs. The main difference from the standard ideal world experiment is that the adversary is allowed to make λ output queries (with possibly different inputs of its choice) in each session. The ideal world execution (parameterized by λ) proceeds as follows.

I. Inputs: P_1 and P_2 obtain a vector of m inputs, denoted $\vec{x}$ and $\vec{y}$ respectively. The adversary is given auxiliary input z, and chooses a party to corrupt. Without loss of generality, we

assume that the adversary corrupts P_2 (when the adversary controls P_1, the roles are simply reversed). The adversary receives the input vector $\vec{y}$ of the corrupted party.

II. Session initiation: The adversary initiates a new session by sending a start-session message to the trusted party. The trusted party then sends $(\mathsf{start\text{-}session}, i)$ to P_1, where i is the index of the session.

III. Honest parties send inputs to trusted party: Upon receiving $(\mathsf{start\text{-}session}, i)$ from the trusted party, honest party P_1 sends (i, x_i) to the trusted party, where x_i denotes P_1's input for session i.

IV. Adversary sends input to trusted party and receives output: Whenever the adversary wishes, it may send a message $(i, \ell, y'_{i,\ell})$ to the trusted party for any $y'_{i,\ell}$ of its choice. Upon sending this pair, it receives back $(i, \ell, f(x_i, y'_{i,\ell}))$ where x_i is the input value that P_1 previously sent to the trusted party for session i. The only limitation is that for any i, the trusted party accepts at most λ tuples indexed by i from the adversary.

Adversary instructs trusted party to answer honest party: When the adversary sends a message of the type $(\mathsf{output}, i, \ell)$ to the trusted party, the trusted party sends $(i, f(x_i, y'_{i,\ell}))$ to P_1, where x_i and $y'_{i,\ell}$ denote the respective inputs sent by P_1 and adversary for session i.

VIII. Outputs: The honest party P_1 always outputs the values $f(x_i, y'_{i,\ell})$ that it obtained from the trusted party. The adversary may output an arbitrary (probabilistic polynomial-time computable) function of its auxiliary input z, input vector $\vec{y}$ and the outputs obtained from the trusted party.

The ideal execution of a function f with security parameter sec, input vectors $\vec{x}$, $\vec{y}$ and auxiliary input z to $\mathcal{S}$, denoted $\text{IDEAL}_{f,\mathcal{S}}(k, \vec{x}, \vec{y}, z)$, is defined as the output pair of the honest party and $\mathcal{S}$ from the above ideal execution.

Definition 1 (λ-Ideal Query Simulator) *Let $\mathcal{S}$ be a non-uniform probabilistic (expected)* PPT *machine representing the ideal-model adversary. We say that $\mathcal{S}$ is a λ-ideal query simulator if it makes at most λ output queries per session in the above ideal experiment.*

REAL MODEL. We now consider the real model in which a real two-party protocol is executed (and there exists no trusted third party). Let f be as above and let Π be a two-party protocol for computing f. Let $\mathcal{A}$ denote a non-uniform probabilistic polynomial-time adversary that controls either P_1 or P_2. The parties run concurrent executions of the protocol Π, where the honest party follows the instructions of Π in all executions. The honest party initiates a new session i with input x_i whenever it receives a start-session message from $\mathcal{A}$. The scheduling of all messages throughout the executions is controlled by the adversary. That is, the execution proceeds as follows: the adversary sends a message of the form (i, msg) to the honest party. The honest party then adds msg to its view of session i and replies according to the instructions of Π and this view. At the conclusion of the protocol, an honest party computes its output as prescribed by the protocol. Without loss of generality, we assume the adversary outputs exactly its entire view of the execution of the protocol.

The real concurrent execution of Π with security parameter k, input vectors $\vec{x}$, $\vec{y}$ and auxiliary input z to $\mathcal{A}$, denoted $\text{REAL}_{\Pi,\mathcal{A}}(k, \vec{x}, \vec{y}, z)$, is defined as the output pair of the honest party and $\mathcal{A}$, resulting from the above real-world process.

SECURITY DEFINITION. Having defined the ideal and real models, we now give our security definition.

Definition 2 (λ-Secure Concurrent Computation in the MIQ Model) *A protocol Π is said to* λ-securely realize *a functionality f under concurrent self composition in the MIQ model if for every real model non-uniform* PPT *adversary $\mathcal{A}$, there exists a non-uniform (expected)* PPT *λ-ideal query simulator $\mathcal{S}$ such that for all polynomials $m = m(k)$, every pair of input vectors $\vec{x} \in X^m$, $\vec{y} \in Y^m$, every $z \in \{0,1\}^*$s,*

$$\{\text{IDEAL}_{f,\mathcal{S}}(k, \vec{x}, \vec{y}, z)\}_{k\in\mathbb{N}} \stackrel{c}{\equiv} \{\text{REAL}_{\Pi,\mathcal{A}}(k, \vec{x}, \vec{y}, z)\}_{k\in\mathbb{N}}$$

2.2 Obfuscation with Tamper-Proof Hardware Tokens

In this work, we use ideas from the area of obfuscation using tamper-proof hardware tokens. In particular, we use the positive result of Goyal et al. [GIS+10] on secure program obfuscation using *stateless* tamper-proof hardware tokens. Below, we give an abstract overview of the scheme of [GIS+10] that we will use in our negative results. We remark that the discussion below hides most of the internal details of the scheme of [GIS+10]. We refer the reader to [GIS+10] for the details of the scheme.

Obfuscation in Token Hybrid Model. In order to obfuscate a circuit C, the sender executes the following steps:

- Sample a token instance $T \leftarrow \mathsf{SampleT}$. The token instance has some secret values hardwired inside it. We will denote the secret values as a key K (that is drawn from some distribution $\mathcal{K}$). In other words, sampling of the token instance just involves to sampling the key K from the distribution $\mathcal{K}$.
- Compute the obfuscated program $\mathcal{O}(C) \leftarrow \mathsf{Obfuscate}(C, K)$

To compute $C(x)$ on any input x, the obfuscated program $\mathcal{O}(C)$ makes t_C queries of various types to the token T, where each q is drawn from some distribution $\mathcal{Q}$. Here, t_C, denoted as the *query parameter* of the obfuscation scheme, is an integer that depends on the size of the circuit C that is obfuscated.

Obfuscating Stateful Programs. The above description is only relevant to obfuscating *stateless* circuits C. We note that it is also possible to obfuscate the programs of *stateful* (or reactive) machines M in the above scheme by using standard techniques. The basic idea is that obfuscated code $\mathcal{O}(M)$ is computed in such a manner that its output on any given input x consists of both $M(x)$ and an *authenticated encryption* of its resultant *state* after the computation of $M(x)$.

We now recall the following security lemma from [GIS+10] (informally stated):

Lemma 1 ([GIS+10]) *Assuming the existence of one-way functions,* ($\mathsf{SampleT}$, $\mathsf{Obfuscate}$) *is a (stateful) program obfuscation scheme in the token-hybrid model with the following properties. For any adversary having the obfuscated program $\mathcal{O}(C)$ along with oracle access to the token, there*

exists an ideal world simulator having only black-box access to the circuit C, such that, the output distribution of the simulator is computationally indistinguishable from that of the adversary.

The Token Functionality. A key idea that is used in our negative results is to implement the working of the hardware token T via a two-party secure computation protocol. To this end, we abstract the working of the token T as the following two-party functionality f_{token}, that we will refer to as the "token functionality".

Denote by f_{token} the token functionality with the key K hardwired inside the description of its circuit. The input and output interface of the functionality f_{token} is described as follows:

Inputs: Party P_1 gets a token key $K \leftarrow \mathcal{K}$ as input, while P_2 gets a query $q \leftarrow \mathcal{Q}$ as input.

Outputs: P_1 gets no output, while P_2 gets $f_{\mathsf{token}}(K; q)$.

Note that f_{token} is a deterministic functionality. We now state a lemma regarding the unpredictability of outputs of f_{token}. We note that this lemma is implicit in [GIS+10].

Lemma 2 (Unpredictability of Output of f_{token} [GIS+10]) *There exists a distribution $\mathcal{Q}$ (with super-logarithmic min-entropy) from which a query q can be sampled with the following properties. Any adversary $\mathcal{A}$, given oracle access to the functionality $f_{token}(K; \cdot)$ (where K is sampled at random from $\mathcal{K}$) with the restriction that it is allowed to query $f_{token}(K; \cdot)$ on any string except q, can output $f_{token}(K; q)$ with only negligible probability.*

Proof Sketch. Observe queries of Type 1 in the obfuscation scheme of [GIS+10]. For a large enough randomly chosen input x, the resulting query q has at least super-logarithmic min-entropy. Note that the output of $f_{\mathsf{token}}(K; q)$ consists of a message-authentication code (MAC) on a message that has full information about q. Hence, the lemma follows from the unforgeability of the MAC scheme. We refer the reader to [GIS+10] for details.

3 Impossibility of Concurrently Secure Computation in the MIQ Model with Black-Box Simulation

In this section, we prove impossibility results for concurrently secure computation in the MIQ model with respect to black-box simulation.

Let $\mathsf{ceil}_1(x) = \lceil x \rceil$. For $y \geq 2$, let $\mathsf{ceil}_y(x)$ be recursively defined as $\mathsf{ceil}_y(x) = \lceil x \cdot \mathsf{ceil}_{y-1}(x) \rceil$. Our general result, stated below, shows a trade-off between the query parameter λ and the round-complexity n of the protocol:

Theorem 2 *There exists a functionality f such that for any $d = d(k)$ and $n = n(k)$ that satisfy $n^d = \mathrm{poly}(k)$, no n-round protocol Π $\lambda = \mathsf{ceil}_d(1 + \frac{1}{n})$-securely realizes f in the MIQ model with respect to black-box simulation.*

Note that $\mathsf{ceil}_d(1+\frac{1}{n}) \geq d+1$. Thus, we obtain the following general corollary when substituting d with λ:

Corollary 1 *There exists a functionality f such that for any $\lambda = \lambda(k)$ and $n = n(k)$ that satisfy $n^\lambda = \mathrm{poly}(k)$, no n-round protocol Π λ-securely realizes f in the MIQ model with respect to black-box simulation.*

By plugging in $n = \text{poly}(k)$ and $\lambda = \mathsf{O}(1)$ above, we get the following as a sub-corollary, ruling out general positive results in the MIQ model when a (black-box) simulator is allowed only a constant number of ideal queries per session:

Corollary 2 *There exists a functionality f that cannot be $\mathsf{O}(1)$-securely realized in the MIQ model with respect to black-box simulation.*

Finally, by plugging in $n = O(1)$ and $d = \log(k)$ in Theorem 2, we obtain the following corollary ruling out constant-round protocols in the MIQ model:

Corollary 3 *There exists a functionality f that cannot be securely realized in the MIQ model by any $O(1)$-round protocol w.r.t. black-box simulation, even if a (black-box) simulator is allowed any (fixed) $\text{poly}(k)$ ideal queries per session.*

We now proceed to the proof of Theorem 2. In fact, we will prove something stronger, as stated below. We first give the following definition:

Definition 3 (λ-special black-box adversary) *Let $\lambda = \lambda(k)$ and $\Pi = (P_1, P_2)$ be a protocol for functionality f. A* PPT *concurrent adversary $\mathcal{A}$ for Π that corrupts party P_2 is said to be* λ-special adversary *if the following holds:*

- *$\mathcal{A}$ outputs* `accept` *with probability 1 in the real-world execution with P_1.*
- *Except with negligible probability, no λ-ideal query black-box simulator can send a query to $\mathcal{A}$ such that it outputs* `accept`*.*

It is easy to see that Theorem 2 is implied by the following theorem:

Theorem 3 *For every $d = d(k)$ and $n = n(k)$-round protocol Π for the f_{token} functionality, if $n^d = \text{poly}(k)$, then there exists a* λ-special adversary *for Π, where $\lambda = \mathsf{ceil}_d(1 + \frac{1}{n})$.*

We prove the above theorem for the f_{token} functionality.

3.1 Proof of Theorem 3

Our proof builds on ideas from Lindell's black-box lower bound for the round complexity of bounded-concurrent self-composition [Lin04]. In what follows, we will follow the notation and terminology used in [Lin04]. Some of the text below is taken from [Lin04].

We will prove the theorem assuming one-way functions. Note that this suffices since secure coin-tossing (even in the stand-alone setting) implies one-way functions. Therefore, if one-way functions do not exist, then it is easy to see that coin-tossing already suffices to prove Theorem 3.

Now, assuming one-way functions, we will prove Theorem 3 for the token functionality f_{token}, as defined in Section 2.2. Our proof will be according to the following outline:

I. First, given any $d = d(k)$ and an n-round protocol Π (s.t. $n^d = \text{poly}(k)$) for f_{token}, we will construct a real-model adversary $\mathcal{A}$ who corrupts party P_2 and interacts with honest P_1 in $m = \frac{n^{d+1}-1}{n-1}$ concurrent sessions with a specific static schedule. By construction, our $\mathcal{A}$ will output `accept` with probability 1 in a real-world execution.

II. Next, we assume for contradiction that with non-negligible probability, there exists a $\lambda(= \mathsf{ceil}_d(1+\frac{1}{n}))$-ideal query black-box simulator $\mathcal{S}$ that manages to send a query to $\mathcal{A}$ such that $\mathcal{A}$ outputs `accept`. We will argue some special properties about $\mathcal{S}$; most notably that there must exist a session at level $i < d$ where $\mathcal{S}$ does not rewind $\mathcal{A}$.

III. Using the above observation, (following standard techniques from [CKL03, Lin04]) we finally construct a new adversary $\mathcal{A}'$ who interacts with an honest P_2 and learns its private input with non-negligible probability. Since this is impossible in the ideal world (from the stand-alone security of Π), we reach a contradiction.

We now proceed to give details. Let us assume for contradiction that there exists an n-round protocol Π that $\lambda(= \mathsf{ceil}_d(1+\frac{1}{n}))$-securely realizes f_{token} under concurrent self-composition. Without loss of generality, we assume that the first message in Π is sent by party P_2.

In the first part of the proof, we will describe a PPT concurrent adversary $\mathcal{A}$ and a specific static message schedule for $m = \frac{n^{d+1}-1}{n-1}$ concurrent executions of Π. Before proceeding further, we first describe how the inputs of the parties P_1 and P_2 are chosen for the m sessions. Recall the description of the token functionality f_{token} from Section 2.2. Let $\mathcal{Q}$ denote a distribution on the token queries with super-logarithmic min-entropy (see Lemma 2). Then, in each session $i \in [m]$, the inputs of P_1 and P_2 are K_i and q_i respectively, where each token key K_i is drawn at random from $\mathcal{K}$, and each q_i is drawn at random from $\mathcal{Q}$.

Part I. Adversary $\mathcal{A}$. Adversary $\mathcal{A}$ is provided as auxiliary input a random string $z \in \{0,1\}^k$, as well as token keys $K_1, \ldots, K_m$ as defined above. $\mathcal{A}$ schedules the protocol messages of the m sessions of Π in the following manner.

Adversarial schedule. Let s denote the first protocol execution initiated by $\mathcal{A}$. We will call it the session at level 0 (i.e., the "outer level"). Now, between every two consecutive rounds of messages in session s, place a new complete protocol session. Let these n nested sessions be denoted as $s_1, \ldots, s_n$. We will call them as sessions at level 1. Now, for every session s_i at level 1, we again place a new complete protocol session between every round of messages in s_i. Note that this creates n^2 sessions denoted as $s_{i,1}, \ldots, s_{i,n}$ at level 2. Repeat this process recursively until we reach level d (i.e., the "inner-most level"), where there are exactly n^d sessions $s_{1,\ldots,1}, \ldots, s_{n,\ldots,n}$. Thus, in total, we have exactly $\frac{n^{d+1}-1}{n-1} = \mathrm{poly}(k)$ sessions. Figure 1 gives a pictorial representation of the adversarial schedule.

$\mathcal{A}$'s strategy. Let $\Pi_1, \ldots, \Pi_m$ denote the m protocol executions, where Π_i denotes the i^{th} session started by $\mathcal{A}$. (That is, $\Pi_1, \ldots, \Pi_m$ is the ordered list of protocol sessions, based on when each session starts.) We now describe the strategy of $\mathcal{A}$ in each of the m protocol sessions.

$\mathcal{A}$ replaces its input q_1 in the first session Π_1 with q_1', where q_1' is sampled from the distribution $\mathcal{Q}$ using auxiliary input z as the randomness. $\mathcal{A}$ starts the execution of Π_1 using q_1' as its input and behaves honestly (running the code of honest P_2). Later, when any session Π_i ($i \in \{2, \ldots, m\}$) starts (as per the schedule), $\mathcal{A}$ collects the partial transcript T_i of the protocol messages (across all sessions) generated so far and applies a pseudo-random function (PRF) F with a random key k to the transcript.[4] It then replaces its input q_i in session Π_i with q_i', where q_i' is sampled from

[4] The key k for the PRF must be included in the auxiliary input of $\mathcal{A}$. We omit it from the description for simplicity of exposition.

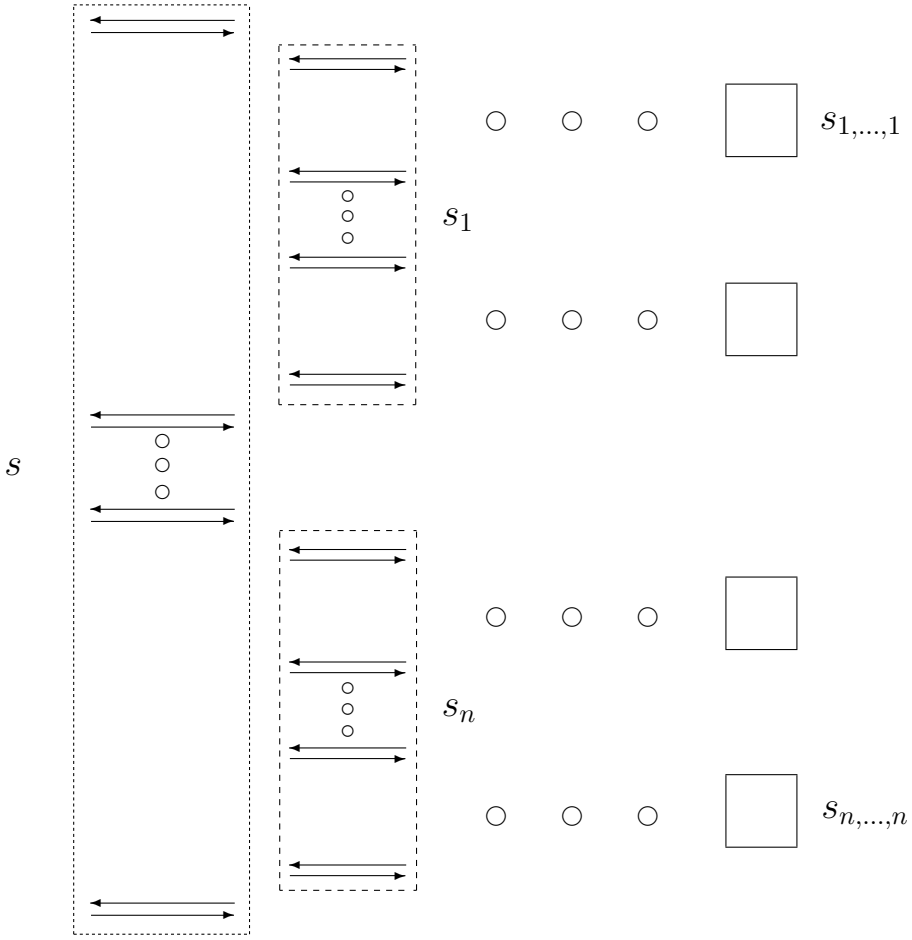

Figure 1: Adversarial schedule for m sessions

distribution $\mathcal{Q}$ using $F_k(T_i)$ as the randomness. $\mathcal{A}$ behaves honestly (by running the code of P_2) in Π_i using q'_i as its input.

In any session $i \in [m]$, if the output σ_i received by $\mathcal{A}$ is such that $\sigma_i \neq f_{\mathsf{token}}(K_i, q'_i)$ (i.e., the output is incorrect[5]) then $\mathcal{A}$ sends $\perp$, outputs all the output values received from the protocol sessions completed so far, and aborts. On the other hand, if $\mathcal{A}$ receives correct answers in all the m sessions, then it outputs `accept`, along with all its inputs and output values from all the sessions.

This completes the description of adversary $\mathcal{A}$. It is easy to see that since $\mathcal{A}$ behaves honestly in each protocol session, it outputs `accept` with probability 1 in a real-world execution with honest P_1.

Part II. Properties of Simulator $\mathcal{S}$. Now, assume for contradiction that with non-negligible probability, there exists a λ-ideal query black-box simulator $\mathcal{S}$ that sends a query to $\mathcal{A}$ such that $\mathcal{A}$ outputs `accept`. Recall from the construction of the adversary $\mathcal{A}$ that it outputs `accept` only when it receives correct outputs in all of the m sessions. In other words, $\mathcal{A}$ outputs `accept` only if $\mathcal{S}$ successfully completes an entire execution of m sessions with $\mathcal{A}$. We will now claim some specific properties about the working of $\mathcal{S}$.

Towards that end, first recall that a black-box simulator only has oracle access to the real world adversary. This oracle represents the next-message function of $\mathcal{A}$ that gets as input the history of the interaction in the form of a query α, and outputs the next message that $\mathcal{A}$ would in an interaction where it sees this history. If $\mathcal{A}$ would abort after receiving a prefix of the messages in an oracle query, then the oracle's response to the query is $\perp$.

We now argue a specific property of all the oracle queries where $\mathcal{A}$ does not abort.

Claim 1 *For every $i \in [m]$, let Φ_i denote the set of all oracle queries sent by $\mathcal{S}$ to $\mathcal{A}$ which includes a full transcript of session Π_i and where $\mathcal{A}$ does not abort. Let T_i denote the transcript prefix before the start of session Π_i. Then, except with negligible probability, at most λ unique prefixes T_i appear in the queries $\alpha \in \Phi_i$.*

Proof. The proof of this claim follows from Lemma 2 and the collision-resistance property of pseudo-random functions. To begin, we first note that except with negligible probability, $\mathcal{S}$ does not make two queries α and α' containing T_i and T'_i such that $T_i \neq T'_i$ but $F_k(T_i) = F_k(T'_i)$. This is easy to see since otherwise we can construct a machine M that distinguishes a pseudo-random function from a random function. Thus, for the remainder of the proof, we make the assumption that such queries α and α' are never made by $\mathcal{S}$.

The claim now follows easily from Lemma 2. Note that $\mathcal{S}$ is allowed to query the ideal functionality at most λ times for session i. Now, suppose for contradiction that with non-negligible probability, there exist $\lambda+1$ different transcript prefixes $T_{i,1}, \ldots, T_{i,\lambda+1}$ that appear in the queries $\alpha \in \Phi_i$. Then, using $\mathcal{S}$ and $\mathcal{A}$, we can construct an adversary B for the token functionality who makes at most λ queries to $f_{\mathsf{token}}(K_i, \cdot)$, but outputs $\lambda + 1$ values $f_{\mathsf{token}}(K_i, q'_{i,1}), \ldots, f_{\mathsf{token}}(K_i, q'_{i,\lambda+1})$, where each $q'_{i,j}$ is sampled from the distribution $\mathcal{Q}$ using randomness $F_k(T_{i,j})$. Since with non-negligible probability, each $F_k(T_{i,j})$ is unique, it follows that with non-negligble probability, each $q'_{i,j}$ is unique as well. Thus, B is a valid adversary for $f_{\mathsf{token}}(K_i, \cdot)$, yielding us a contradiction to Lemma 2. ■

Next, we claim that $\mathcal{S}$ must extract $\mathcal{A}$'s input in each execution of Π_i that has a unique prefix T_i, and where $\mathcal{A}$ does not abort. (Note that there may be multiple executions of Π_i due to rewinding by $\mathcal{S}$ in sessions "enclosing" Π_i.)

[5]Recall that f_{token} is a determinisitic functionality.

Claim 2 *For every $i \in [m]$, in every execution of Π_i that has a unique prefix T_i and where $\mathcal{A}$ does not abort, except with negligible probability, $\mathcal{S}$ must query the ideal functionality f_{token} on $\mathcal{A}$'s input q_i' in that execution of Π_i.*

Proof. Recall from the construction of $\mathcal{A}$ that it checks whether its output σ_i at the end of an execution of Π_i with prefix T_i is the correct value $f_{\mathsf{token}}(K_i, q_i')$, where q_i' is $\mathcal{A}$'s input in that execution sampled from $\mathcal{Q}$ using randomness $F_k(T_i)$. (Note that $\mathcal{A}$ can checks the correctness of output since it receives the token key K_i as auxiliary input.) Then, it follows easily from Lemma 2 that $\mathcal{S}$ must query the ideal functionality f_{token} on q_i' since otherwise, we can construct an adversary B that has oracle access to $f_{\mathsf{token}}(K_i, \cdot)$ and with non-negligble probability, correctly predicts the value $f_{\mathsf{token}}(K_i, q_i')$ without ever querying $f_{\mathsf{token}}(K_i, \cdot)$ on input q_i'. The details follow in a similar manner to the proof of previous claim and are omitted. ■

Thus, it follows from the previous claim that $\mathcal{S}$ extracts $\mathcal{A}$'s input in each protocol session where $\mathcal{A}$ does not abort. This brings us to our final and most important claim about $\mathcal{S}$. Let Ψ_s denote the subset of all the queries from $\mathcal{S}$ to $\mathcal{A}$ that contain any message in session s. We claim the following:

Claim 3 *There exists a session s^* at level $\ell < d$ such that:*

- *There does not exist a pair of queries $\alpha, \alpha' \in \Psi_j$ that contain the same transcript prefix T_{s^*}, but contain two different round r messages msg_r, msg_r' in session s^* for some $r \in [n]$.*
- *$\mathcal{S}$ queries the ideal functionality f_{token} on $\mathcal{A}$'s input in s^* (as defined by the prefix T_{s^*}).*

Proof. (Sketch) Note that the above lemma essentially states that there exists an execution of session s^* at level $j < d$ where $\mathcal{S}$ extracts $\mathcal{A}$'s input without rewinding $\mathcal{A}$. Then, let us assume for contradiction that $\mathcal{S}$ extracts $\mathcal{A}$'s input in each session $i \in [m]$ by rewinding $\mathcal{A}$. Further suppose that $\mathcal{S}$ succeeds in extracting $\mathcal{A}$'s input in each session by rewinding only once per session.[6]

Now, let us view the schedule as an n-ary tree where the root at level 0 denotes the outer session and the n^d nodes at level d denote the innermost sessions. Then, first note that since the outer session is rewound at least once, it follows that the n sessions at level 1 are executed in total $n+1$ times. In other words, there exists a session (say) s_i at level 1 that is executed at least $\mathsf{ceil}_1(1+\frac{1}{n})$ times, each time with a different prefix (this is because the last message of the prefix must be different in order for the rewinding in session s to be successful). Now, it follows from Claim 2 that $\mathcal{S}$ will need to extract $\mathcal{A}$'s input in each of these executions in order to complete them successfully. Then, focusing on the sub-tree with root s_i, we have that the n child nodes of s_i are executed in total $(n+1)\mathsf{ceil}_1(1+\frac{1}{n})$ times. In other words, there exists a child node of s_i at level 2 that is executed at least $\mathsf{ceil}_2(1+\frac{1}{n})$ times, each time with a different prefix. Continuing this argument inductively, it follows that at every level j, there exists at least one session that is executed $\mathsf{ceil}_j(1+\frac{1}{n})$ times. As a result, we obtain that there exist at least one session at level d that is executed $\mathsf{ceil}_d(1+\frac{1}{n})$ times, every time with a different prefix. However, this contradicts Claim 1. ■

Part III. Adversary $\mathcal{A}$' for breaking stand-alone security of P_2. We will now construct an adversary $\mathcal{A}$' that interacts with an honest P_2 in an execution of Π and obtains P_2's input query $q \in \mathcal{Q}$ with non-negligible probability to contradict the stand-alone security of f_{token}.

[6] It is easy to see that more rewindings by $\mathcal{S}$ only facilitate our proof.

Recall that it follows from Claim 3 that there exists a session s^* at level $j < d$ such that in every execution of s^*, $\mathcal{S}$ extracts $\mathcal{A}$'s input without rewinding. Since $\mathcal{A}$ honestly plays the role of party P_2 in each session (see description of $\mathcal{A}$), it follows that $\mathcal{S}$ must be able to extract the input of an honest P_2 as well. Then, adversary $\mathcal{A}$' works as follows. It simply runs the black-box simulator $\mathcal{S}$ and emulates the role of P_2 in each session $s \neq s^*$ by using random queries $q \in \mathcal{Q}$ as inputs for P_2. Further, $\mathcal{A}$ samples keys $K_s \leftarrow \mathcal{K}$ for each session $s \neq \mathsf{s}^*$ and uses them to answer the ideal functionality queries of $\mathcal{S}$. Finally, $\mathcal{A}'$ forwards the messages of $\mathcal{S}$ in (the first execution of) session s^* to the external honest party P_2, and returns P_2's responses to $\mathcal{S}$. $\mathcal{S}$ runs and at some point, queries the ideal functionality on the input of P_2 in (the forwarded execution of) session s^*. $\mathcal{A}$ receives this query from $\mathcal{S}$ and thus obtains the input of the external party P_2, which is a contradiction.

Full details of the construction of $\mathcal{A}'$ follow in the same manner as in [Lin04], and are therefore omitted.

4 Full Impossibility of Concurrently Secure Computation in the MIQ Model

Recall that the impossibility results presented in Section 3.1 are only relevant to black-box simulation. In this section, we present full impossibility results for concurrently secure computation in the multiple ideal query model, i.e., we now rule out non-black-box simulation as well. Interestingly, we rely on the black-box impossibility results from Section 3.1 in order to obtain our full impossibility results.

More concretely, we present a technique to "compile" our impossibility results w.r.t. black-box simulation into full impossibility results, thus ruling out non-black-box simulation as well. We now state our main theorem of this section:

Theorem 4 *Let $\lambda = \lambda(k)$ and Π be any protocol for the f_{token} functionality. If there exists a* PPT *λ-special black-box adversary $\mathcal{A}$ for Π, then there exists a* PPT *concurrent adversary $\mathcal{B}$ for Π whose view cannot be simulated by any (potentially non-black-box)* PPT *λ-ideal query simulator.*

Recall that we proved our impossibility results in Section 3.1 by constructing a λ-special black-box adversary $\mathcal{A}$ for any n-round protocol for f_{token} with $\lambda = \mathsf{ceil}_d(1 + \frac{1}{n})$, where d is such that $n^d = \text{poly}(k)$. Thus, by combining Theorem 3 with the above theorem, we immediately obtain the following corollaries:

Corollary 4 *For any $d = d(k)$ and $n = n(k)$ that satisfy $n^d = \text{poly}(k)$, there exists no n-round protocol Π that λ-securely realizes f_{token} in the MIQ model for $\lambda = \mathsf{ceil}_d(1 + \frac{1}{n})$.*

Corollary 5 *For any $\lambda = \lambda(k)$ and $n = n(k)$ that satisfy $n^\lambda = \text{poly}(k)$, there exists no n-round protocol Π that λ-securely realizes f_{token} in the MIQ model.*

Corollary 6 *The functionality f_{token} cannot be $\mathsf{O}(1)$-securely realized in the MIQ model.*

Corollary 7 *The functionality f_{token} cannot be λ-securely realized in the MIQ model by any $O(1)$-round protocol for any $\lambda = \text{poly}(k)$.*

We now proceed to give a formal proof of Theorem 4.

4.1 Proof of Theorem 4

Let $\lambda = \lambda(k)$ and Π be any protocol for the f_{token} functionality. Then, given any λ-special black-box adversary $\mathcal{A}$ for Π, we now show how to construct a λ-special adversary $\mathcal{B}$ for Π. We use ideas from obfuscation using stateless tamper-proof hardware tokens [GIS+10] in order to show this transformation.

We start by giving the outline of the proof:

I. We first consider a toy experiment involving three parties, namely, Alice, Bob and David, where Alice and David are honest parties, and Bob is the adversary. (This will be the case throughout our proof.) In this experiment, Alice and Bob interact in multiple ideal world executions of the functionality f_{token}. At the same time, Bob and David are involved in concurrent real-world executions of Π where Bob and David follow the same scheduling of messages as defined by adversary $\mathcal{A}$. Furthermore, the adversary Bob is allowed to *reset* David at any point during their interaction.

David, who has a secret input secret is instructed to reveal secret to Bob if all the executions of Π are completed "successfully". (We define this precisely later.) The goal of Bob is to successfully complete its interaction with David and learn the value secret. Then, by relying on our black-box impossibility result, we will show that no adversarial Bob can succeed in learning secret, except with negligible probability.

II. In the second step, we will use ideas from [GIS+10] to obfuscate David's program in the f_{token}-hybrid model and give it as an auxiliary input to Bob. This results in a scenario where Alice and Bob are the only parties, who interact in multiple ideal world executions of f_{token}. We will show that in this *ideal world experiment*, no adversarial Bob cannot learn secret, except with negligible probability.

III. We finally consider the *real world experiment*, which is the same as step II, except that all the ideal world invocations of f_{token} are now replaced with real-world executions of protocol Π. We will show that in the real experiment, an adversarial Bob can learn secret with probability 1.

Our adversary $\mathcal{B}$ is simply the adversary Bob in step III. Note that from steps II and III, it immediately follows that $\mathcal{B}$ is a PPT concurrent adversary for Π whose view cannot be simulated by any λ-ideal query simulator, thus yielding us the proof of Theorem 4.

We now proceed to describe each of the above three steps in details. We first setup some notation that is common to the three steps.

Notation. Let m be the number of sessions that adversary $\mathcal{A}$ schedules for protocol Π. Let $K_1, \ldots, K_m$ denote a set of token keys, where each K_i is drawn at random from the distribution $\mathcal{K}$. Let $q_1, \ldots, q_m$ be a set of query strings for the token functionality, where each q_i is drawn at random from the distribution D (as defined in Lemma 2). Finally, let secret be a random string in $\{0,1\}^k$.

(Toy) Experiment I: Restating the Black-Box Impossibility Result. Consider the following scenario involving three parties, namely, Alice, Bob and David. Alice is given an input vector $(K_1, \ldots, K_m)$, while David is given an input vector $((K_1, q_1), \ldots, (K_m, q_m); \mathsf{secret})$. Here, the input

values are chosen in the manner as described above. We describe the interaction between Alice and Bob, and Bob and David, separately.

Interaction between Alice and Bob. Alice and Bob are interacting in m ideal world executions of the functionality f_{token}, where Alice plays the role of P_1 using input vector $(K_1, \ldots, K_m)$ and Bob plays the role of P_2 using any inputs of its choice. In each of these m ideal world executions, the adversary Bob is allowed to query the token functionality λ times using any inputs of its choice.

Interaction between Bob and David. At the same time, Bob and David are interacting in m real-world concurrent executions of protocol Π, where Bob plays the the role of P_1 with any inputs of its choice and David plays the role of P_2 by simply running the code of the λ-special adversary $\mathcal{A}$.[7]

Bob and David are instructed to assume the identities of Alice and Bob, respectively, in these sessions. The messages of the m sessions follow the same schedule as defined by the adversary $\mathcal{A}$. Furthermore, the adversary Bob is allowed to *reset* David at any point during their interaction. That is, at any point during their interaction, Bob can choose to "rewind" David to an earlier state and create new threads of execution.

In each session $i \in [m]$, David verifies whether his output $y_i = f_{\mathsf{token}}(K_i, q_i)$ (where q_i is the input of David in that session, chosen in the same manner as $\mathcal{A}$ would); if this is not the case, then David aborts the interaction with Bob and outputs $\perp$ (i.e., David does not continue any remaining sessions with Bob). If *all* of the m sessions are completed successfully, then David sends secret to Bob (as the only additional message outside of the m concurrent executions of Π).

We now claim the following lemma:

Lemma 3 *Bob outputs* secret *in Experiment I with negligible probability.*

The proof of the above lemma follows in a straightforward way from the observation that if Bob outputs secret with non-negligible probability, then there exists a λ-ideal query black-box simulator for the λ-special adversary $\mathcal{A}$, which contradicts our assumption on $\mathcal{A}$. This simulator is simply the description of the party Bob.

Experiment II: Ideal World. We now eliminate the party David from the above toy experiment by using techniques from [GIS+10]. Note that eliminating David results only in interactions between Alice and Bob, which is indeed our desired setting of concurrent self-composition. Very roughly, in order to eliminate David, we would like to *obfuscate* David's next message function and give it as auxiliary input to Bob. However, recall that general program obfuscation is impossible in the plain model [BGI+01]. Our key idea then is to use positive results on secure program obfuscation using tamper-proof hardware tokens, and adapt them to our setting. In particular, we will use the positive results of Goyal et al. [GIS+10] on secure program obfuscation using a stateless hardware token that implements the f_{token} functionality. We now give more details. Some of the notation used below is as defined in Section 2.2

Eliminating David. Consider the next message function NMF of party David as described in Experiment I. Sample a key $K \leftarrow \mathcal{K}$. Compute the obfuscation of NMF in the f_{token}-hybrid model, where f_{token} has the key K hardwired. The resulting program, denoted as $\mathcal{O}(\mathsf{NMF}) \leftarrow \mathsf{Obfuscate}(K, \mathsf{NMF})$, is given as auxiliary input to Bob. The key K, on the other hand, is given as an additional input to Alice.

[7] In particular, if $\mathcal{A}$ chooses to ignore the inputs $q_1, \ldots, q_m$ and choose fresh inputs "on-the-fly", then David follows the same strategy.

Ideal World Experiment. Experiment II, or in other words, the ideal world experiment is defined in the same manner as Experiment I, except that we eliminate the party David in the manner as described above. Note that in order to evaluate the program $\mathcal{O}(\mathsf{NMF})$ on any input, Bob would need to answer the queries q of $\mathcal{O}(\mathsf{NMF})$ to the token functionality $f_{\mathsf{token}}(K, \cdot)$. In particular, recall from Section 2.2 that the obfuscated code $\mathcal{O}(C)$ for a program C makes t_C queries to the token functionality, where t_C (referred to as the query parameter) depends on the size of the circuit C. Then, in the ideal world experiment, we have Alice and Bob engage in $t_{\mathsf{NMF}} \cdot m \cdot n$ additional ideal executions of f_{token}, where t_{NMF} is the query parameter for the obfuscation scheme of [GIS+10] as determined by the circuit description of NMF, m is the number of sessions that adversary $\mathcal{A}$ schedules, and n is the round complexity of protocol Π. In each of these $t_{\mathsf{NMF}} \cdot m \cdot n$ executions, Alice uses the key K as input, while Bob is allowed to use any inputs of its choice. Further, in each of these $t_{\mathsf{NMF}} \cdot m \cdot n$ ideal executions, Bob is allowed to query the token functionality λ times using any inputs of its choice.

Thus, overall the ideal world experiment between Alice and Bob consists of the following:

- m "main" ideal executions of f_{token}, where in each session $i \in [m]$, Alice uses key K_i as input, while Bob is allowed to use any inputs of its choice.
- $t_{\mathsf{NMF}} \cdot m \cdot n$ "auxiliary" ideal executions of f_{token}, where in each session, Alice uses key K as input, while Bob is allowed to use any inputs of its choice.

Further, as explained above, in *each* of the $m + t_{\mathsf{NMF}} \cdot m \cdot n$ ideal executions of f_{token}, Bob is allowed to query the ideal functionality λ times using any inputs of its choice.[8]

We now claim the following:

Lemma 4 *Bob outputs* secret *in Experiment II with negligible probability.*

Proof. The proof of the above lemma comes from the following simple observation. It follows from the security of the obfuscation scheme of Goyal et al. (see Lemma 1) that giving the obfuscated code $\mathcal{O}(\mathsf{NMF})$ and the $t_{\mathsf{NMF}} \cdot m \cdot n$ ideal executions of f_{token} as described above (or rather any polynomial number of ideal executions of f_{token} with key K) to the adversary Bob is the same as giving Bob oracle access to the next-message function of party David. Thus, the view of Bob is computationally indistinguishable in Experiments I and II. Then, the above lemma follows immediately from Lemma 3. ■

Experiment III: Real World. We now describe the final experiment, which is the real world experiment between Alice and Bob. This experiment is essentially the same as Experiment II, except that each ideal world execution of f_{token} is now replaced with a real world execution of protocol Π between Alice and Bob. In more detail, the interaction between Alice and Bob consists of the following:

- m "main" executions of protocol Π, where the messages of the m executions are scheduled in the same manner as defined by the adversary $\mathcal{A}$. In each $i \in [m]$ main session, Alice uses the keys K_i as her input.
- $t_{\mathsf{NMF}} \cdot m \cdot n$ "auxiliary" executions of protocol Π that are scheduled by Bob in the manner, as described below. In each of these auxiliary sessions, Alice uses key K as her input.

[8] Note that this is the case since we are considering λ-secure concurrent computation.

In more detail, the real world experiment is described by the following programs:

Alice's program: Alice is given input the keys $K_1, \ldots, K_n$ for the m "main" sessions, and key K for the $t_{\mathsf{NMF}} \cdot m \cdot n$ "auxiliary" sessions.

Alice behaves honestly according to the protocol Π and responds honestly to all protocol invocations made by Bob by using the code of P_1.

Bob's program: Bob is given as auxiliary input the obfuscated program $\mathcal{O}(\mathsf{NMF})$, where NMF is the next-message function of David (as described above). Let secret be the secret value hardwired in $\mathcal{O}(\mathsf{NMF})$.

For $i = 1, \ldots, m \cdot n$, do:

1. Upon receiving the i^{th} message from Alice in m main sessions, say a_i, suspend (temporarily) the ongoing session. Run the code $\mathcal{O}(\mathsf{NMF})$ on input a_i.[9] Whenever $\mathcal{O}(\mathsf{NMF})$ makes a query q, start a new auxiliary session of Π with Alice, and run the code of P_2 honestly using input q. Since $\mathcal{O}(\mathsf{NMF})$ makes t_{NMF} different queries, in total, t_{NMF} auxiliary sessions of Π are executed sequentially by Bob.

2. If $i < m \cdot n$, then on receiving the output d_i from $\mathcal{O}(\mathsf{NMF})$ outputs, resume the suspended "main" session and send d_i to Alice as the response to a_i. Otherwise, output the value $d_i = \mathsf{secret}$.

Lemma 5 *Bob outputs the value* secret *in Experiment III with probability* 1.

The proof of the above lemma follows immediately from the description of Bob.

Completing the Proof of Theorem 4. The adversary $\mathcal{B}$ is simply the adversary Bob in Experiment III as described above. The proof of Theorem 4 then follows immediately from Lemma 4 and 5.

References

[AGJ+12] Shweta Agrawal, Vipul Goyal, Abhishek Jain, Manoj Prabhakaran, and Amit Sahai. New impossibility results on concurrently secure computation and a non-interactive completeness theorem for secure computation. In *CRYPTO*, 2012.

[BCL+05] Boaz Barak, Ran Canetti, Yehuda Lindell, Rafael Pass, and Tal Rabin. Secure computation without authentication. In *CRYPTO*, pages 361–377, 2005.

[BCNP04] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *FOCS*, pages 186–195, 2004.

[9] Note that the input to $\mathcal{O}(\mathsf{NMF})$ would also include the authenticated state information that $\mathcal{O}(\mathsf{NMF})$ would have output earlier. We exclude this from the description for simplicity of notation.

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

[BPS06] Boaz Barak, Manoj Prabhakaran, and Amit Sahai. Concurrent non-malleable zero knowledge. In *FOCS*, pages 345–354, 2006.

[BS05] Boaz Barak and Amit Sahai. How to play almost any mental game over the net - concurrent composition via super-polynomial simulation. In *FOCS*, pages 543–552, 2005.

[Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–147, 2001.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *CRYPTO*, pages 19–40, 2001.

[CHK+05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In *EUROCRYPT*, pages 404–421, 2005.

[CKL03] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In *EUROCRYPT*, pages 68–86, 2003.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503, 2002.

[CLP10] Ran Canetti, Huijia Lin, and Rafael Pass. Adaptive hardness and composable security in the plain model from standard assumptions. In *FOCS*, pages 541–550, 2010.

[CPS07] Ran Canetti, Rafael Pass, and Abhi Shelat. Cryptography from sunspots: How to use an imperfect reference string. In *FOCS*, pages 249–259, 2007.

[DNS98] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. In *STOC*, pages 409–418, 1998.

[GGJ13] Vipul Goyal, Divya Gupta, and Abhishek Jain. What information is leaked under concurrent composition? In *CRYPTO*, 2013.

[GGJS12] Sanjam Garg, Vipul Goyal, Abhishek Jain, and Amit Sahai. Concurrently secure computation in constant rounds. In *EUROCRYPT*, 2012.

[GIS+10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In *TCC*, pages 308–326, 2010.

[GJO10] Vipul Goyal, Abhishek Jain, and Rafail Ostrovsky. Password-authenticated session-key generation on the internet in the plain model. In *CRYPTO*, pages 277–294, 2010.

[GKOV12] Sanjam Garg, Abishek Kumarasubramanian, Rafail Ostrovsky, and Ivan Visconti. Impossibility results for static input secure computation. In *CRYPTO*, 2012.

[GL01] Oded Goldreich and Yehuda Lindell. Session-key generation using human passwords only. In *CRYPTO*, pages 408–432, 2001.

[GL03] Rosario Gennaro and Yehuda Lindell. A framework for password-based authenticated key exchange. In *EUROCRYPT*, pages 524–543, 2003.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In *STOC*, pages 218–229, 1987.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.

[Goy12] Vipul Goyal. Positive results for concurrently secure computation in the plain model. In *FOCS*, 2012.

[GS09] Vipul Goyal and Amit Sahai. Resettably secure computation. In *EUROCRYPT*, 2009.

[Kat07] J. Katz. Universally composable multi-party computation using tamper-proof hardware. In *Advances in Cryptology — Eurocrypt 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 115–128. Springer, 2007.

[KL11] Dafna Kidron and Yehuda Lindell. Impossibility results for universal composability in public-key models and with fixed inputs. *J. Cryptology*, 24(3):517–544, 2011.

[KOY01] Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *EUROCRYPT*, pages 475–494, 2001.

[KP01] Joe Kilian and Erez Petrank. Concurrent and resettable zero-knowledge in poly-logarithm rounds. In *STOC*, pages 560–569, 2001. Preliminary full version published as cryptology ePrint report 2000/013.

[Lin03] Yehuda Lindell. General composition and universal composability in secure multi-party computation. In *FOCS*, pages 394–403, 2003.

[Lin04] Yehuda Lindell. Lower bounds for concurrent self composition. In *TCC*, pages 203–222, 2004.

[Lin08] Yehuda Lindell. Lower bounds and impossibility results for concurrent self composition. *J. Cryptology*, 21(2):200–249, 2008.

[LP12] Huijia Lin and Rafael Pass. Black-box constructions of composable protocols without set-up. In *CRYPTO*, pages 461–478, 2012.

[LPV09] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. A unified framework for concurrent security: universal composability from stand-alone non-malleability. In *STOC*, pages 179–188. ACM, 2009.

[MP06] Silvio Micali and Rafael Pass. Local zero knowledge. In *STOC*, pages 306–315, 2006.

[MPR06] Silvio Micali, Rafael Pass, and Alon Rosen. Input-indistinguishable computation. In *FOCS*, pages 367–378, 2006.

[NP99] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *STOC*, pages 245–254, 1999.

[NP06] Moni Naor and Benny Pinkas. Oblivious polynomial evaluation. *SIAM J. Comput.*, 35(5):1254–1281, 2006.

[Pas03] Rafael Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In *EUROCRYPT*, pages 160–176, 2003.

[Pas12] Rafael Pass. Personal communication. 2012.

[PR08] Manoj Prabhakaran and Mike Rosulek. Cryptographic complexity of multi-party computation problems: Classifications and separations. In *CRYPTO*, pages 262–279, 2008.

[PRS02] Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *FOCS*, pages 366–375, 2002.

[PS04] Manoj Prabhakaran and Amit Sahai. New notions of security: achieving universal composability without trusted setup. In *STOC*, pages 242–251, 2004.

[RK99] Ransom Richardson and Joe Kilian. On the concurrent composition of zero-knowledge proofs. In *EUROCRYPT*, pages 415–431, 1999.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *FOCS*, pages 162–167, 1986.