Dynamic Security: A Realistic Approach to Adaptive Security With Applications to Strong FaF Security

Bar Alon
Department of Computer Science
Georgetown University,
Washington, DC, USA
alonbar08@gmail.com

Naty Peter*
University of Ottawa
Ottawa, ON, Canada
naty@post.bgu.ac.il

May 27, 2025

Abstract

Secure multiparty computation allows multiple parties to jointly compute a function while maintaining security even in the presence of malicious adversaries. There are two types of adversaries in the literature: static adversaries, which choose the parties to corrupt before the protocol begins; and adaptive adversaries, which can corrupt parties during the execution of the protocol based on the messages exchanged by the parties. While adaptive security provides a more robust security guarantee, it may require too much in certain scenarios. Indeed, the adversary must allocate some of its resources to corrupt the parties; however, certain parties might be more susceptible to corruption, for instance, if they have not updated their operating system to the latest version.

To address this, we introduce a new security notion called *dynamic security*. Here, adversaries may corrupt new parties *during and after* the protocol's execution, but *cannot choose* targets based on the messages. A protocol is said to be (t, h)-dynamically secure if it is possible to simulate any adversary that can corrupt up to t parties during the execution and h thereafter. Dynamic security provides meaningful security for many real-world scenarios. Moreover, it circumvents known lower bounds on the communication complexity of adaptive security, allowing for more efficient protocols such as committee-based ones, which would be insecure against adaptive adversaries.

We further explore dynamic security and establish the following results.

1. We show a surprising connection between dynamic security and the seemingly unrelated notion of security with friends and foes (FaF security), introduced by Alon et al. (CRYPTO 2020), which aims to protect honest parties not only from adversaries but also against other honest parties. The notion of (t, h)-*strong FaF security* strengthens this by requiring the simulatability of the joint view of any t malicious parties alongside any h honest parties to be indistinguishable from their real-world view. We show that (t, h)-dynamic security and (t, h)-strong FaF security are equivalent.
2. We consider the feasibility of (t, h)-dynamic security and show that every n-party functionality can be computed with computational (t, h)-dynamic security (with guaranteed output delivery) if and only if $2t + h < n$. By our previous result, this also solves an open problem left by Alon et al. on the feasibility of strong FaF security.

*Part of the work conducted while at the University of Toronto. Partially supported by the National Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant RGPIN-2023-05006.

Contents

1 Introduction **1**
- 1.1 Our Contributions . . . 2
- 1.2 Our Techniques . . . 6
- 1.3 Related Works . . . 8
- 1.4 Organization . . . 9

2 Preliminaries **9**
- 2.1 Security Model: Security With Friends and Foes . . . 9
- 2.2 Security Model: Adaptive Security . . . 14

3 A New Security Definition: Dynamic Security **16**
- 3.1 Dynamic Security-With-Abort . . . 19

4 Equivalence of Dynamic Security and Strong FaF Security **20**

5 Characterizing Computational Dynamic Security **23**
- 5.1 Strong FaF Security-With-Semi-Identifiable-Abort . . . 24
- 5.2 Generating Correlated Randomness with Strong FaF Security . . . 25
- 5.3 Putting it all Together . . . 28

Bibliography **29**

1 Introduction

Secure multiparty computation (MPC) allows a set of mutually distrustful parties to perform a common task. Such computations should satisfy various security properties, such as correctness, privacy, fairness, independence of inputs, and even guaranteed output delivery.[1] Moreover, security should hold even when the parties interact with a malicious entity that controls a subset of the parties. The two common types of adversaries that are considered in the literature are *static* and *adaptive* adversaries. A static adversary chooses the subset of parties to corrupt *before* the protocol starts. The seminal works of [24, 19, 5, 11, 23] show how to construct protocols that are secure against any static adversary. An adaptive adversary can corrupt new parties *during* the execution of the protocol. It was first considered by Beaver and Haber [4] and Canetti et al. [7], and it offers a more realistic security notion than static security. The seminal result of [9] shows the construction of an adaptively secure protocol in the common reference string model.

Achieving security against adaptive adversaries is naturally harder. Indeed, there are several examples of protocols that are secure against static adversaries but are insecure against adaptive ones [7, 8, 20, 16, 3, 13], and an example of a functionality that cannot be computed with adaptive security [21].[2] Additionally, lower bounds on the communication complexity of adaptively secure protocols show they are less efficient than statically secure ones [17].

While adaptive security may be stronger, we argue that in certain scenarios it may require too much: The adversary must assign some of its resources to try and corrupt an arbitrary party, which may be an inefficient task. One way to model this is by bounding the number of corruptions the adaptive adversary can make. However, a more realistic model is where some parties are *easier* to corrupt. This can happen for several reasons. For example, certain parties can be more prone to participate in such corrupted activity in exchange for a reward, or several parties can be deceived into being corrupted by the adversary without their will. The latter can be illustrated by several real-world scenarios, e.g., where some parties did not update their operating system to the latest version, making them an easier target; or where the adversary tries to corrupt using a phishing attack and some parties are more likely to click the link. In these scenarios, it is natural to require security against an adversary that can still corrupt parties during the execution of the protocol; however, it cannot choose who to corrupt based on the protocol's execution.

Achieving security against such adversaries may allow us to bypass certain lower bounds and impossibility results of adaptive security, while still guaranteeing a more realistic security notion than static security. Indeed, consider a protocol where the parties randomly sample a small committee, share their inputs with the parties in the committee, and let the committee perform the computation. As noted by Canetti et al. [7], an adaptive adversary can corrupt the entire committee, thus making the protocol insecure. In fact, Garay et al. [17] showed that there is no adaptively secure protocol with sublinear communication. However, note that such a protocol may still be secure against adversaries who do not get to choose who they corrupt. This is due to the fact that the choice of corruption is independent of the random choices made by the parties.

[1] Formally, security is defined via the real vs. ideal world paradigm, where the real-world interaction is required to emulate an ideal one.

[2] Ishai et al. [21] showed that there is a randomized functionality that cannot be computed against adversaries that can corrupt all parties. This is only a concern when considering adaptive adversaries as it is important for composition.

1.1 Our Contributions

Our main contribution is the introduction of a new security notion, which we call *dynamic security*,[3] to realistically capture the scenarios discussed above. That is, it captures security against adversaries that can corrupt parties during the protocol's execution, however, they cannot choose who they corrupt and when. We then show a surprising connection between dynamic security and the seemingly unrelated notion of security with friends and foes [1], which aims to protect honest parties from other honest parties (in addition to protecting them from the adversary). Finally, we characterize the security threshold that allows to compute any functionality with dynamic security.

1.1.1 Defining Dynamic Security

Our main contribution is defining dynamic security, which can be viewed as an intermediate security notion that lies between static and adaptive security. Recall that in static security, the adversary chooses the set of corrupted parties prior to the beginning of the protocol, whereas in adaptive security, the adversary can arbitrarily corrupt additional parties during the protocol, possibly utilizing information gained from previous rounds in order to choose the next parties to be corrupted. Therefore, when acknowledging the limitations of the above two security definitions, our main motivation for defining dynamic security is to capture security against more realistic adversaries than static ones (as accomplished by adaptive security), while also obtaining more efficient protocols and for broader classes of functionalities than the ones obtained by adaptively secure protocols, hopefully overcome some lower bounds imposed by this type of requirement.

Similarly to previous security notions, we define dynamic security using the real vs. ideal world paradigm. In this work, we only consider the simpler stand-alone model, where there are no other procedures that occur concurrently with the execution of the protocol. We first describe the real world, which aims to describe the interaction with an adversary that can corrupt parties during the protocol's execution (as in adaptive security), but cannot choose the parties to corrupt nor the round in which the parties will be corrupted. We then define the ideal world, where the parties interact via an incorruptible trusted party that does the computation on the parties' behalf. Security is then defined by requiring that any adversary in the real world can be simulated by an adversary in the ideal world. We consider the ideal world for defining both full security (also known as guaranteed output delivery) and security-with-abort. Roughly, the former notion requires that the honest parties always obtain the output, while the latter relaxes this requirement and allows the adversary to learn the output and prevent the honest parties from receiving it.

Real-world execution. Roughly speaking, we model the adversary $\mathcal{A}$ as follows. The information about the parties to be corrupted at every round is provided to $\mathcal{A}$ by an external entity $\mathcal{Z}$, called the environment, which interacts with the adversary in a similar way to how it is done in adaptively secure protocols. However, as opposed to adaptive security, in dynamic security, the environment determines the parties to be corrupted at every round prior to the running of the protocol. When $\mathcal{Z}$ instructs $\mathcal{A}$ to corrupt a party, the adversary corrupts it, obtains its internal

[3] The term dynamic security was used in previous works to refer to what is now commonly known as adaptive security [4, 23]. We chose to use the term dynamic as adaptive insinuates that the adversary can change its structure based on what occurs in the protocol, while dynamic only insinuates that the structure can be changed based on sources outside of the adversary's control. The term dynamic was also used by [12, 14] in the context of fluid MPC.

state (which includes its input, randomness, and messages it received so far),[4] and continues with the execution of the protocol. Finally, to allow for the composition of secure protocols, we further let the environment choose which parties the adversary will corrupt *after* the protocol terminates, in what is referred to as the *post-execution corruption (PEC)* phase.

We next provide more details. Fix an n-party r-round protocol. At the beginning of the protocol, the environment first generates a *corruption list* $\mathcal{L} = (\rho_1, \rho_2, \ldots, \rho_n)$ corresponding to the parties $\mathsf{P}_1, \mathsf{P}_2, \ldots, \mathsf{P}_n$, where each ρ_i indicates if party P_i is going to be corrupted and at which round. That is, we have that $\rho_i \in [r] \cup \{\mathsf{PEC}, \mathsf{honest}\}$ for every $i \in [n]$; at each round $\rho \in [r]$, the adversary corrupts all parties P_i such that $\rho_i = \rho$, and after the execution of the protocol the adversary corrupts all parties P_i such that $\rho_i = \mathsf{PEC}$. If $\rho_i = \mathsf{honest}$, then P_i is an honest party not corrupted by the adversary at any stage of the protocol. During the protocol's execution, the corrupted parties follow the instructions of the adversary from the time they have been corrupted. The protocol terminated with the adversary outputting a function of its view, and the honest parties outputting whatever the protocol instructs them to output.

After the protocol terminates, the PEC phase begins. The environment receives the output of the adversary and the outputs of the honest parties, and instructs the adversary to corrupt all parties P_i such that $\rho_i = \mathsf{PEC}$. The adversary corrupts the parties and computes a message to send to $\mathcal{Z}$. Finally, the environment outputs some value based on its information and the final message received by the adversary.

If the environment generates a corruption list such that the number of parties corrupted during the execution of a protocol is bounded by t and the number of parties corrupted after the termination of a protocol is bounded by h, then we say that the environment is (t, h)-limited.[5]

Ideal-world execution for full security. We next describe the ideal world for full security, where the computation is done by a trusted party. Similarly to the real world, the execution starts with the environment generating a corruption list $\mathcal{L} = (\rho_1, \rho_2, \ldots, \rho_n)$. There will be three distinct corruption stages in the ideal world. The first corruption stage occurs before any interaction with the trusted party is done. In this stage, the adversary and the environment interact in several iterations. In each iteration, the environment may instruct the adversary to corrupt a party P_i such that $\rho_i \in [r]$. In such a case, the adversary corrupts P_i and obtains its input.

After the first corruption stage ends, the parties send their inputs to the trusted party: all honest parties send the inputs they hold, while the corrupted parties send a value from their input domain (possibly their actual input), as dictated by the adversary. The trusted party then computes the output and sends it to the parties (and the adversary). Now, the second corruption stage begins. The only difference from the first stage is that now, upon corrupting a party, the adversary also obtains the output it received from the trusted party. After the second stage is over, each honest party outputs whatever it received from the trusted party, and the adversary outputs a function of its ideal-world view (aiming to simulate the output of a real-world adversary). Finally, the PEC phase begins, with the adversary and the environment further interacting similarly to the real-world

[4] Formally, the adversary should also receive additional auxiliary information, which corresponds to information the party has from previous executions. This is important to allow for the composition of secure protocols. However, to simplify this introduction, we ignore such details.

[5] It is more common to provide a single upper bound on the total number of parties corrupted by the adversary (as done in all papers on adaptive security we are aware of). However, distinguishing between corruptions done during the execution of the protocol and corruptions done during the PEC phase will allow us to clearly present the connection between dynamic security and security with friends and foes [1]. See Theorem 1.2 below.

PEC phase.

Ideal-world execution for security-with-abort. The ideal world for security-with-abort proceeds similarly to the ideal world for full security, with the following changes. (1) When the trusted party computes the function, it sends the output only to the adversary. (2) During the second corruption stage, whenever the adversary corrupts a new party, the trusted party sends to the adversary the corrupted party's output. This is important since in a real-world execution, some of the honest parties might learn the output while some do not. (3) After the second corruption stage is over, the adversary can instruct the trusted party whether it should send the honest parties their output or instruct them to abort.

Security definition. We are now ready to define dynamic security. We say that a protocol computes some multiparty function with (t, h)-dynamic security if for every adversary and every (t, h)-limited dynamic environment, there is a simulator interacting in the above ideal world for full security, such that the output of the dynamic environment in both the real and ideal executions are indistinguishable. If there exists a simulator in the ideal world for security-with-abort, then we say that the protocol is (t, h)-dynamically secure-with-abort. We refer the reader to Section 3 for a formal description of the security definition.

Remark 1.1. *Observe that the definition of dynamic security, while weaker than adaptive security, still captures some of the concerns that motivate adaptive security. Indeed, the internal memory of honest parties cannot be considered "safe" in the dynamic security setting. However, unlike adaptive security, protocols can still be more efficient in the dynamic security setting. Consider the following example for computing an arbitrary n-party functionality f. The n parties first elect a random committee of size $\omega(\log n)$ using Feige's committee election protocol [15], then share their inputs with the parties in the committee, and let the committee run an adaptively secure protocol to compute f. The protocol is clearly insecure against adaptive adversaries since the adversary can corrupt the entire committee. However, in dynamic security, the set of parties to be corrupted by the adversary is chosen before the committee is sampled. Therefore, the probability that the adversary corrupts the entire committee is negligible, hence the protocol is secure.*

1.1.2 Equivalence to Strong Friends and Foes Security

Friends and foes (FaF) security was defined by Alon et al. [1] to capture the privacy of honest parties against other honest parties. This is formalized by considering two adversaries that do not collude. One adversary is malicious and may deviate from the protocol in any way, while the other is semi-honest and follows the protocol. In contrast to adaptive and dynamic security, FaF security considers static adversaries, which means that both (disjoint) sets of corrupted parties and semi-honest parties are determined before the execution of the protocol.

Alon et al. [1] presented two flavors of FaF security. The weaker notion requires that any malicious adversary can be simulated, and that any semi-honest simulator can be simulated while interacting with the malicious adversary. In strong FaF security, it is required that the joint distribution of the views of the two adversaries is indistinguishable from the joint distribution of the simulated views in the ideal world. In more detail, (t, h)-strong FaF security requires that for any real-world malicious adversary $\mathcal{A}_{\mathsf{mal}}$ corrupting at most t parties there is an ideal-world malicious simulator $\mathcal{S}_{\mathsf{mal}}$, and for any semi-honest adversary $\mathcal{A}_{\mathsf{sh}}$ corrupting at most h of the remaining parties

there exists an ideal-world semi-honest simulator $\mathcal{S}_{\mathsf{sh}}$, such that $(\text{VIEW}^{\mathsf{real}}_{\mathcal{A}_{\mathsf{mal}}}, \text{VIEW}^{\mathsf{real}}_{\mathcal{A}_{\mathsf{sh}}}, \text{OUT}^{\mathsf{real}})$ is indistinguishable from $(\text{VIEW}^{\mathsf{ideal}}_{\mathcal{S}_{\mathsf{mal}}}, \text{VIEW}^{\mathsf{ideal}}_{\mathcal{S}_{\mathsf{sh}}}, \text{OUT}^{\mathsf{ideal}})$, where $\text{OUT}^{\mathsf{real}}$ and $\text{OUT}^{\mathsf{ideal}}$ denote the output in the real-world and ideal-world, respectively, of the parties not corrupted by the malicious adversary and simulator (in particular, this protects the semi-honest parties from the malicious adversary).

For our first result, we show an equivalence between strong FaF secure protocols and dynamically secure protocols. This result holds for computational, statistical, and perfect security, and also for security-with-abort.

Theorem 1.2 (Informal)**.** *Let π be a protocol computing some functionality f. Then π computes f with (t,h)-dynamic security (respectively, security-with-abort) if and only if π computes f with (t,h)-strong FaF security (respectively, security-with-abort).*

By using the positive and negative results of Alon et al. [1] for statistical and perfect strong FaF security, Theorem 1.2 readily implies the following corollary for dynamic security.

Corollary 1.3 (Informal)**.** *If $t+h < n/2$, then any n-party functionality can be computed with statistical (t,h)-dynamic security. Otherwise, there is an n-party functionality that cannot be computed with statistical (t,h)-dynamic security.*

If $3t + 2h < n$, then any n-party functionality can be computed with perfect (t,h)-dynamic security. Otherwise, there is an n-party functionality that cannot be computed with perfect (t,h)-dynamic security.

The equivalence we show between dynamic security and strong FaF security can be useful in both directions, as we next discuss. On one hand, when considering strong FaF secure protocols, this equivalence gives additional motivation for this security notion and emphasizes its significance. It shows that strong FaF security captures more settings than what was suggested by the original definition, which can be translated to the notion of dynamic security. This makes strong FaF security a more versatile and useful security notion.

On the other hand, when considering dynamically secure protocols, the above equivalence provides us with a simpler way of proving dynamic security by proving strong FaF security instead. Indeed, in strong FaF security, the sets of semi-honest parties and corrupted parties are fixed prior to the beginning of the protocol. This is in contrast to dynamic security, where the adversary corrupts new parties during the protocol.

1.1.3 Characterization of Dynamic Security

Given a (t,h)-limited dynamic environment and a set of n parties, we show that any n-party functionality can be computed with computational dynamic security if and only if $2t + h < n$.

Theorem 1.4 (Informal)**.** *If $2t+h < n$, then under several cryptographic assumptions, any n-party functionality can be computed with computational (t,h)-dynamic security. Otherwise, there is an n-party functionality that cannot be computed with computational (t,h)-dynamic security.*

See Theorem 5.1 for the formal statement and the list of cryptographic assumptions. Following the equivalence between dynamically secure and strong FaF secure protocols, Theorem 1.4 solves an open problem left by Alon et al. [1] about the feasibility of computational strong FaF secure protocols.

1.2 Our Techniques

We next provide an overview of our techniques. We start by proving Theorem 1.2, i.e., by showing that dynamically secure protocols are also strong FaF secure and vice versa. We then use this equivalence to prove Theorem 1.4.

Proof of Theorem 1.2. Recall that we want to prove an equivalence between (t,h)-dynamic and (t,h)-strong FaF security for any n-party protocol computing some functionality f.

First, we take a protocol that is (t,h)-dynamically secure and show that it is also (t,h)-strong FaF secure. Fix a malicious adversary $\mathcal{A}_{\mathsf{mal}}$ corrupting a set of parties $\mathcal{I}$ of at most t parties, and a semi-honest adversary $\mathcal{A}_{\mathsf{sh}}$ corrupting a set of parties $\mathcal{H}$ (disjoint from $\mathcal{I}$) of at most h parties. Intuitively, $\mathcal{A}_{\mathsf{mal}}$ and $\mathcal{A}_{\mathsf{sh}}$ can be perfectly emulated by a single adversary $\mathcal{A}_{\mathsf{dyn}}$ corrupting the parties in $\mathcal{I}$ in the beginning of the protocol, and corrupting the parties in $\mathcal{H}$ after it terminates. Since these corruptions are made independently of the protocol's execution, they can be generated by a (t,h)-limited dynamic environment $\mathcal{Z}$. In more detail, we let $\mathcal{A}_{\mathsf{dyn}}$ run $\mathcal{A}_{\mathsf{mal}}$ during the protocol's execution, and during the PEC phase, we let it run $\mathcal{A}_{\mathsf{sh}}$ and send its output to $\mathcal{Z}$.

What left to show are two simulators $\mathcal{S}_{\mathsf{mal}}$ and $\mathcal{S}_{\mathsf{sh}}$ for $\mathcal{A}_{\mathsf{mal}}$ and $\mathcal{A}_{\mathsf{sh}}$, respectively. First, as we assume (t,h)-dynamic security, there is an ideal-world simulator $\mathcal{S}_{\mathsf{dyn}}$ for $\mathcal{A}_{\mathsf{dyn}}$ such that the output of $\mathcal{Z}$ is indistinguishable from its output in the real world. The idea for the malicious simulator $\mathcal{S}_{\mathsf{mal}}$ is to let it run $\mathcal{S}_{\mathsf{dyn}}$ while taking the role of the environment $\mathcal{Z}$. That is, whenever $\mathcal{Z}$ instructs $\mathcal{S}_{\mathsf{dyn}}$ to corrupt a party, $\mathcal{S}_{\mathsf{mal}}$ sends the party's input (and output in the second corruption stage) to $\mathcal{S}_{\mathsf{dyn}}$. Note that this can be done since $\mathcal{Z}$ only instructs to corrupt the parties that are already in control of $\mathcal{S}_{\mathsf{mal}}$. For the semi-honest simulator $\mathcal{S}_{\mathsf{sh}}$, we let it output the message sent by $\mathcal{S}_{\mathsf{dyn}}$ to $\mathcal{Z}$ during the post-execution corruption phase. Now, recall that in the definition of strong FaF security, the semi-honest simulator receives the ideal-world view of the malicious simulator (which includes its inputs, output received from the trusted party, and randomness). Therefore, the output of $\mathcal{S}_{\mathsf{sh}}$ is consistent with the output of $\mathcal{S}_{\mathsf{mal}}$, as required by strong FaF security.

We now prove the second direction. That is, we show that if a protocol π is (t,h)-strong FaF secure, then it is also (t,h)-dynamically secure. Let r denote the number of rounds in π. Fix a real-world dynamic adversary $\mathcal{A}_{\mathsf{dyn}}$. We would like to first emulate $\mathcal{A}_{\mathsf{dyn}}$ using a pair of non-colluding adversaries and use their simulators to construct a simulator for $\mathcal{A}_{\mathsf{dyn}}$. However, no single pair of such adversaries can emulate $\mathcal{A}_{\mathsf{dyn}}$, since the corruptions are chosen by an environment. Instead, our idea is to define such a pair for every possible corruption list $\mathcal{L} = (\rho_1, \rho_2, \ldots, \rho_n)$, let the simulator reconstruct the corruption list by interacting with the environment in the first corruption stage, and then run the simulators assumed to exist by the strong FaF security guarantee. However, note that since the environment can only instruct the simulator to corrupt parties P_i such that $\rho_i \in [r]$, it does not have enough information to run the malicious simulator corresponding to $\mathcal{L}$. To overcome this, we define the malicious adversary for every *partial corruption list*, which is defined as $\mathcal{L}^* = (\rho_1^*, \rho_2^*, \ldots, \rho_n^*)$, where $\rho_i^* \in ([r] \cup \{*\})$ for every $i \in [n]$. Note that the dynamic simulator can reconstruct the partial corruption list by interacting with the environment in the first corruption stage. We next give more details.

The malicious adversary $\mathcal{A}_{\mathsf{mal},\mathcal{L}^*}$ emulates $\mathcal{A}_{\mathsf{dyn}}$ in every round $\rho \in [r]$ of the protocol as follows: First, $\mathcal{A}_{\mathsf{mal},\mathcal{L}^*}$ takes the role of the environment and instructs $\mathcal{A}_{\mathsf{dyn}}$ to corrupt all parties P_i such that $\rho_i^* = \rho$. Then, for every corrupted party P_i with $\rho_i^* > \rho$ (that is, the parties yet to be corrupted by $\mathcal{A}_{\mathsf{dyn}}$ at round ρ), it computes its honest messages as specified by the protocol and sends them to the honest parties. In return, it receives messages from the honest parties, and sends them to $\mathcal{A}_{\mathsf{dyn}}$,

together with the messages computed for the corrupted parties P_i with $\rho_i^* > \rho$. Finally, $\mathcal{A}_{\mathsf{mal},\mathcal{L}^*}$ receives from $\mathcal{A}_{\mathsf{dyn}}$ the messages for the newly corrupted parties of the previous step and sends them to the honest parties, and the next round begins. After the protocol terminates, $\mathcal{A}_{\mathsf{mal},\mathcal{L}^*}$ broadcasts its view. Next, for a corruption list $\mathcal{L} = (\rho_1, \rho_2, \ldots, \rho_n)$, define the semi-honest adversary $\mathcal{A}_{\mathsf{sh},\mathcal{L}}$ that corrupts the parties in $\mathcal{H} := \{\mathsf{P}_i : \rho_i = \mathsf{PEC}\}$ and emulates the post-execution corruption phase of $\mathcal{A}_{\mathsf{dyn}}$: After receiving the broadcast message of the malicious adversary, it sends it to $\mathcal{A}_{\mathsf{dyn}}$, which responds with the message its sends to the dynamic environment in the PEC phase.

By our security assumption, there are ideal-world simulators $\mathcal{S}_{\mathsf{mal},\mathcal{L}^*}$ and $\mathcal{S}_{\mathsf{sh},\mathcal{L}}$ for $\mathcal{A}_{\mathsf{mal},\mathcal{L}^*}$ and $\mathcal{A}_{\mathsf{sh},\mathcal{L}}$, respectively (this holds for every possible $\mathcal{L}$ whose associated partial corruption list is $\mathcal{L}^*$). We next use them to construct the simulator $\mathcal{S}_{\mathsf{dyn}}$ for $\mathcal{A}_{\mathsf{dyn}}$. In the first corruption stage, $\mathcal{S}_{\mathsf{dyn}}$ interacts with the dynamic environment $\mathcal{Z}$ for r rounds. The environment instructs $\mathcal{S}_{\mathsf{dyn}}$ to corrupt the parties in $\mathcal{I} := \{\mathsf{P}_i : \rho_i \in [r]\}$. The simulator then runs $\mathcal{S}_{\mathsf{mal},\mathcal{L}^*}$ for the corruption list $\mathcal{L}^* = (\rho_1^*, \rho_2^*, \ldots, \rho_n^*)$ defined as follows: $\rho_i^* = \rho$ if the environment instructed to adversary to corrupt P_i at round ρ, and $\rho_i^* = *$ otherwise. Note that since the first corruption stage had r rounds, there is no second corruption stage. Finally, in the post-execution phase, the dynamic environment instructs the simulator to corrupt the parties in $\mathcal{H} := \{\mathsf{P}_i : \rho_i = \mathsf{PEC}\}$. We let $\mathcal{S}_{\mathsf{dyn}}$ run $\mathcal{S}_{\mathsf{sh},\mathcal{L}}$, compute its output, and send it to the environment.

By construction, for every PPT environment $\mathcal{Z}$, the outputs of $\mathcal{S}_{\mathsf{dyn}}$ and the honest parties are indistinguishable from the outputs of $\mathcal{A}_{\mathsf{mal},\mathcal{L}^*}$ and the honest parties, where $\mathcal{L}$ is generated by $\mathcal{Z}$. By the assumption of strong FaF security, the same holds for the output of the semi-honest adversary conditioned on these values. This implies that the output of $\mathcal{Z}$ is indistinguishable.

Proof of Theorem 1.4. By Theorem 1.2 it suffices to prove the theorem for (t, h)-strong FaF security. That is, we construct a (t, h)-strong FaF secure protocol for any n-party functionality, when $2t + h < n$; and show that if $2t + h \geq n$ then there is an n-party functionality that cannot be computed with (t, h)-strong FaF security. Since the latter was proved by Alon et al. [1],[6] we only need to consider the case when $2t + h < n$.

Our starting point is the result by Melissaris et al. [22], who constructed a (t, h)-strong FaF secure protocol in the correlated randomness setting. Therefore, it suffices to construct a strong FaF secure protocol for sampling correlated randomness. The idea is as follows. Consider the protocol of Garg and Sahai [18]. They showed how to compute any n-party functionality with adaptive security-with-abort against adversaries that can corrupt up to $n - 1$ parties. We note that their protocol in fact achieves a stronger security requirement: in case of an abort, the honest parties all agree on a pair of parties such that at least one of them is corrupted. We call this security notion *security-with-semi-identifiable-abort*. Since adaptive security implies strong FaF security [1], their protocol is (t, h)-strong FaF secure-with-semi-identifiable-abort.

Using the above observation, we propose the following protocol for sampling correlated randomness. We first present a protocol that is secure against fail-stop adversaries (i.e., the corrupted parties follow the protocol faithfully but may abort prematurely) rather than malicious ones. A protocol tolerating malicious adversaries is given below. The parties start by executing the Garg and Sahai protocol to compute $(t + h + 1)$-out-of-n Shamir's secret sharing of the output. If the computation followed through, then the assumption that $2t + h < n$ implies that the uncorrupted

[6] They in fact showed that this holds for the weaker security notion of (t, h)-FaF security.

parties, consisting of at least

$$n - t \geq 2t + h + 1 - t = t + h + 1$$

parties, can reconstruct the output. Otherwise, the parties have the identity of a pair of parties such that at least one of them is corrupted. The parties remove them from the computation and restart, with the shares of the removed parties to be held shared among the remaining parties. If the computation follows through then the parties send to the removed parties their shares. If the protocol aborted, the parties remove two more parties and continue as before. Since $2t + h < n$, this process will end with either an honest majority or with one of the intermediate computations successfully generating the output. To obtain a protocol against malicious adversaries, the parties also compute signatures of the shares.

As an example, consider the case where $t = 2$, $h = 1$, and $n = 6$. If the adversary aborts in the first iteration, the remaining 4 parties compute a 3-out-4 Shamir's secret sharing of the output. Since at most one of them is malicious, if the computation is followed through, then the malicious party cannot prevent the *removed* honest party from reconstructing its output. If the adversary aborts in the second iteration as well, then only two uncorrupted parties remain (i.e., both follow the protocol). They will compute a 2-out-of-2 secret sharing of the output of all other parties and send them the shares.

1.3 Related Works

Adaptive security was studied substantially, with many proposed constructions admitting adaptive security [23, 4, 7, 9, 18, 10]. Since adaptive security implies dynamic security, all of these results apply to our setting as well.

Canetti et al. [8] were the first to study the connection between adaptive and static security, showing equivalences and separations under different settings. For example, against malicious adversaries, they showed that even in the three-party setting with perfect security, there is a protocol that is secure against two static corruptions, but is insecure against 2 adaptive corruptions. Since two static corruptions is equivalent to $(2, 0)$-strong FaF security, which in turn is equivalent to $(2, 0)$-dynamic security, this separates adaptive security from dynamic security in the perfect security setting.

Canetti et al. [8] further showed that any statically secure protocol satisfying additional basic security requirements is also secure against adaptive adversaries, albeit with *inefficient* simulation. Asharov, Cohen, and Shochat [3] showed that the inefficient simulator is inherent, assuming the existence of one-way permutations. They also proved that the BGW protocol [5] is secure against adaptive (semi-honest) adversaries with efficient simulation.

The notion of FaF security was introduced by Alon, Omri, and Paskin-Cherniavsky [1], and later studied in [22, 2]. Alon et al. [1] further provided feasibility and impossibilities for FaF security. Most relevant to our work are their impossibility results, which also apply to dynamic security. In particular, we have the following two results. (1) For all t,h, and n such that $2t + h \geq n$ there is a functionality that cannot be computed with (t, h)-dynamic security. (2) For every n there is an n-party functionality that cannot be computed with $(1, 1)$-dynamic security in two rounds. FaF security was further studied by Alon, Beimel, and Omri [2]. They provided a family of 3-party functionalities that cannot be computed with $(1, 1)$-dynamic security in a logarithmic number of rounds.

1.4 Organization

In Section 2, we provide the required preliminaries and the definitions of FaF and adaptive security. Then, in Section 3, we formally define our new notion of dynamic security. Later on, in Section 4, we show the equivalence between dynamic security and strong FaF security. Finally, in Section 5, we characterize the parameters that allow for computational dynamic security of every functionality.

2 Preliminaries

We use calligraphic letters to denote sets, uppercase for random variables, lowercase for values, and bold characters to denote vectors. For $n \in \mathbb{N}$, let $[n] = \{1, 2 \dots n\}$. For a set $\mathcal{S}$ we write $s \leftarrow \mathcal{S}$ to indicate that s is selected uniformly at random from $\mathcal{S}$. Given a random variable (or a distribution) X, we write $x \leftarrow X$ to indicate that x is selected according to X. PPT stands for probabilistic polynomial time.

A function $\mu(\cdot)$ is called negligible if for every positive polynomial $p(\cdot)$ and all sufficiently large n, it holds that $\mu(n) < 1/p(n)$. For a vector $\mathbf{v}$ of dimension n, we write v_i for its i^{th} component, and for $\mathcal{S} \subseteq [n]$ we write $\mathbf{v}_{\mathcal{S}} = (v_i)_{i \in \mathcal{S}}$. For a randomized function (or an algorithm) f we write $f(x)$ to denote the random variable induced by the function on input x, and write $f(x; r)$ to denote the value when the randomness of f is fixed to r.

A *distribution ensemble* $X = \{X_{a,n}\}_{a \in \mathcal{D}_n, n \in \mathbb{N}}$ is an infinite sequence of random variables indexed by $a \in \mathcal{D}_n$ and $n \in \mathbb{N}$, where $\mathcal{D}_n$ is a domain that might depend on n. The statistical distance between two finite distributions is defined as follows.

Definition 2.1. *The statistical distance between two finite random variables X and Y is*

$$\mathrm{SD}(X, Y) = \frac{1}{2} \sum_a \left| \Pr[X = a] - \Pr[Y = a] \right|.$$

Two ensembles $X = \{X_{a,n}\}_{a \in \mathcal{D}_n, n \in \mathbb{N}}$ and $Y = \{Y_{a,n}\}_{a \in \mathcal{D}_n, n \in \mathbb{N}}$ are said to be statistically close, if there exists a negligible function $\mu(\cdot)$, such that for all n and $a \in \mathcal{D}_n$,

$$\mathrm{SD}(X_{a,n}, Y_{a,n}) \leq \mu(n).$$

If $\mu(n) = 0$ for all $n \in \mathbb{N}$ then we say that X and Y are identically distributed.

Computational indistinguishability is defined as follows.

Definition 2.2. *Let $X = \{X_{a,n}\}_{a \in \mathcal{D}_n, n \in \mathbb{N}}$ and $Y = \{Y_{a,n}\}_{a \in \mathcal{D}_n, n \in \mathbb{N}}$ be two ensembles. We say that X and Y are computationally indistinguishable, denoted $X \overset{\mathrm{C}}{\equiv} Y$, if for every PPT distinguisher D, there exists a negligible function $\mu(\cdot)$, such that for all $n \in \mathbb{N}$ and $a \in \mathcal{D}_n$,*

$$\left| \Pr[\mathsf{D}(X_{a,n}) = 1] - \Pr[\mathsf{D}(Y_{a,n}) = 1] \right| \leq \mu(n).$$

2.1 Security Model: Security With Friends and Foes

In this section, we define security with friends and foes (FaF security) [1]. This notion strengthens the classical definition of security by imposing privacy restrictions on (subsets of) honest parties, even in the presence of malicious behavior by other parties. The definition also prevents the

adversary from leaking private information of one subset of parties to another subset of parties, even though neither subset is under its control. This is modeled by having two adversaries that do not collude, where one adversary is malicious and the other is semi-honest.

The definition follows the standard *real vs. ideal* paradigm for defining security. Intuitively, the security notion is defined by describing an ideal functionality, in which both the corrupted and non-corrupted parties interact with a trusted entity. A real-world protocol is deemed secure if an adversary in the real world cannot cause more harm than an adversary in the ideal world. The classical security definition (which is only concerned with the malicious adversary) captures this by showing that an ideal-world adversary (simulator) can simulate the full view of the real-world malicious adversary. For FaF security, we further require that the view of a subset of the uncorrupted parties can be simulated in the ideal world (including the interaction with the adversary). We define FaF security against *static* adversaries, namely, where the set of corrupted parties and semi-honest parties are fixed before the execution of the protocol begins.

The FaF Real Model

An n-party protocol π is defined by a set of n interactive probabilistic polynomial-time Turing machines $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$. Each Turing machine (party) holds at the beginning of the execution the common security parameter 1^κ, a private input, and random coins.

We consider the interaction to be over a synchronous network. That is, the execution proceeds in rounds: each round consists of a *send phase* (where parties send their messages for this round) followed by a *receive phase* (where they receive messages from other parties). We consider a fully connected point-to-point network, where every pair of parties is connected by a communication line. We also consider the *secure-channels* model, where the communication lines are assumed to be ideally private (and thus the adversary cannot read or modify messages sent between two honest parties). Additionally, we assume the parties have access to a broadcast channel, allowing each party to faithfully send the same message to all other parties.

An adversary is another interactive Turing machine. It starts the execution with an input that contains the identity of the corrupted parties, their inputs, and an additional auxiliary input $z \in \{0,1\}^*$. For the FaF setting, we will only consider static adversaries that can choose the subset of parties to corrupt prior to the execution of the protocol. At the end of the protocol's execution, the adversary outputs some function of its view (which consists of its random coins, its auxiliary input, the input of the corrupted parties, and the messages it sees during the execution of the protocol, and specifically, including possibly non-prescribed messages sent by a malicious adversary).

We consider two adversaries. The first adversary we consider is a malicious adversary $\mathcal{A}_{\mathsf{mal}}$ that controls a subset $\mathcal{I} \subseteq \mathcal{P}$ of the parties. The adversary has access to the full view of the corrupted parties. The adversary may instruct the corrupted parties to deviate from the protocol in any way it chooses. The adversary can send messages (even if not prescribed by the protocol) to any uncorrupted party in every round of the protocol and can do so after all messages for this round are sent. The adversary can also send messages to the uncorrupted parties *after* the protocol is terminated. The adversary is also given an auxiliary input z_{mal}.

The second adversary is a semi-honest adversary $\mathcal{A}_{\mathsf{sh}}$ that controls a subset $\mathcal{H} \subseteq \mathcal{P} \setminus \mathcal{I}$ of the remaining parties (for the sake of clarity, we will only refer to the parties in $\mathcal{I}$ as corrupted, the parties in $\mathcal{P} \setminus \mathcal{I}$ as uncorrupted, the parties in $\mathcal{H}$ as semi-honest, and the remaining parties as honest). Similarly to $\mathcal{A}_{\mathsf{mal}}$, this adversary also has access to the full view of the parties it controls.

However, $\mathcal{A}_{\mathsf{sh}}$ *cannot* instruct the parties to deviate from the prescribed protocol in any way, but may try to infer information about the remaining honest parties (i.e., those in $\mathcal{P} \setminus (\mathcal{I} \cup \mathcal{H})$), given its view in the protocol. This adversary is given an auxiliary input z_{sh}.

We next define the global view of the real world. For inputs $\mathbf{x} = (x_1, \ldots, x_n)$, security parameter $\kappa \in \mathbb{N}$, and adversaries $\mathcal{A}_{\mathsf{mal}}$ and $\mathcal{A}_{\mathsf{sh}}$ controlling the parties in $\mathcal{I} \subseteq \mathcal{P}$ and $\mathcal{H} \subseteq \mathcal{P} \setminus \mathcal{I}$, and holding auxiliary inputs z_{mal} and z_{sh}, respectively, we define the following. Let $\mathcal{A}_{\mathsf{FaF}} = (\mathcal{A}_{\mathsf{mal}}, \mathcal{A}_{\mathsf{sh}})$, let $z_{\mathsf{FaF}} = (z_{\mathsf{mal}}, z_{\mathsf{sh}})$, let $\text{MAL-VIEW}^{\mathsf{real},\mathsf{FaF}}_{\pi,\mathcal{A}_{\mathsf{mal}}}(\kappa, \mathbf{x}, z_{\mathsf{mal}})$ denote the output of $\mathcal{A}_{\mathsf{mal}}$, let $\text{SH-VIEW}^{\mathsf{real},\mathsf{FaF}}_{\pi,\mathcal{A}_{\mathsf{FaF}}}(\kappa, \mathbf{x}, z_{\mathsf{FaF}})$ denote the output of $\mathcal{A}_{\mathsf{sh}}$, and let $\text{OUT}^{\mathsf{real},\mathsf{FaF}}_{\pi,\mathcal{A}_{\mathsf{mal}}}(\kappa, \mathbf{x}, z_{\mathsf{mal}})$ denote the outputs of the uncorrupted parties (i.e., those in $\mathcal{P} \setminus \mathcal{I}$), during an execution of π when running alongside $\mathcal{A}_{\mathsf{mal}}$.

We denote the global view in the real model by

$$\text{REAL}^{\mathsf{FaF}}_{\pi,\mathcal{A}_{\mathsf{FaF}}}(\kappa, \mathbf{x}, z_{\mathsf{FaF}}) = \left(\text{MAL-VIEW}^{\mathsf{real},\mathsf{FaF}}_{\pi,\mathcal{A}_{\mathsf{mal}}}(\kappa, \mathbf{x}, z_{\mathsf{mal}}),\ \text{SH-VIEW}^{\mathsf{real},\mathsf{FaF}}_{\pi,\mathcal{A}_{\mathsf{FaF}}}(\kappa, \mathbf{x}, z_{\mathsf{FaF}}),\ \text{OUT}^{\mathsf{real},\mathsf{FaF}}_{\pi,\mathcal{A}_{\mathsf{mal}}}(\kappa, \mathbf{x}, z_{\mathsf{mal}})\right).$$

It will be convenient to denote

$$\text{MAL-REAL}^{\mathsf{FaF}}_{\pi,\mathcal{A}_{\mathsf{mal}}}(\kappa, \mathbf{x}, z_{\mathsf{mal}}) = \left(\text{MAL-VIEW}^{\mathsf{real},\mathsf{FaF}}_{\pi,\mathcal{A}_{\mathsf{mal}}}(\kappa, \mathbf{x}, z_{\mathsf{mal}}),\ \text{OUT}^{\mathsf{real},\mathsf{FaF}}_{\pi,\mathcal{A}_{\mathsf{mal}}}(\kappa, \mathbf{x}, z_{\mathsf{mal}})\right),$$

i.e., the projection of $\text{REAL}^{\mathsf{FaF}}_{\pi,\mathcal{A}_{\mathsf{FaF}}}(\kappa, \mathbf{x}, z_{\mathsf{FaF}})$ to the output of $\mathcal{A}_{\mathsf{mal}}$ and the uncorrupted parties' output (those in $\mathcal{P} \setminus \mathcal{I}$), and denote

$$\text{SH-REAL}^{\mathsf{FaF}}_{\pi,\mathcal{A}_{\mathsf{FaF}}}(\kappa, \mathbf{x}, z_{\mathsf{FaF}}) = \left(\text{SH-VIEW}^{\mathsf{FaF}}_{\pi,\mathcal{A}_{\mathsf{FaF}}}(\kappa, \mathbf{x}, z_{\mathsf{FaF}}),\ \text{OUT}^{\mathsf{real},\mathsf{FaF}}_{\pi,\mathcal{A}_{\mathsf{mal}}}(\kappa, \mathbf{x}, z_{\mathsf{mal}})\right),$$

i.e., the projection of $\text{REAL}^{\mathsf{FaF}}_{\pi,\mathcal{A}_{\mathsf{FaF}}}(\kappa, \mathbf{x}, z_{\mathsf{FaF}})$ to the output of $\mathcal{A}_{\mathsf{sh}}$ and the uncorrupted parties. When π is clear from the context, we will remove it for brevity.

The FaF Ideal Model

We next describe the interaction in the *FaF security ideal model*, which specifies the requirements for FaF security. We consider an ideal computation with *guaranteed output delivery* (also referred to as full security), where a trusted party T performs the computation on behalf of the parties, and the ideal-world malicious adversary cannot abort the computation. This computation is parameterized by a (potentially randomized) n-party functionality $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ to compute. An ideal computation of f on input $\mathbf{x} = (x_1, \ldots, x_n)$ with security parameter κ in the presence of a malicious adversary (a simulator) $\mathcal{S}_{\mathsf{mal}}$ corrupting $\mathcal{I}$, and a semi-honest adversary $\mathcal{S}_{\mathsf{sh}}$ corrupting $\mathcal{H}$, proceeds as follows.

Inputs: Each party P_i holds 1^κ and a private input $x_i \in \{0,1\}^*$. The adversaries $\mathcal{S}_{\mathsf{mal}}$ and $\mathcal{S}_{\mathsf{sh}}$ are given auxiliary inputs $z_{\mathsf{mal}} \in \{0,1\}^*$ and $z_{\mathsf{sh}} \in \{0,1\}^*$, respectively, and the private input of every party controlled by them. The trusted party T holds 1^κ.

Parties send inputs: Each uncorrupted party $\mathsf{P}_i \in \mathcal{P} \setminus \mathcal{I}$ sends x_i as its input to T. For each corrupted party, the malicious adversary $\mathcal{S}_{\mathsf{mal}}$ sends to T a value from its domain. In case the adversary does not send any input, the trusted party replaces its input with a default value. Write $(x'_1, \ldots, x'_n)$ for the tuple of inputs received by the trusted party.

The trusted party performs computation: The trusted party T samples a random string rnd, computes $\mathbf{y} = (y_1, \ldots, y_n) = f(x'_1 \ldots, x'_n; \mathsf{rnd})$, and sends y_i to party P_i for every $i \in [n]$. If $\mathsf{P}_i \in \mathcal{I}$ then $\mathcal{S}_{\mathsf{mal}}$ receives y_i, and if $\mathsf{P}_i \in \mathcal{H}$ then $\mathcal{S}_{\mathsf{sh}}$ receives y_i.

The malicious adversary sends its (ideal-world) view: $\mathcal{S}_{\mathsf{mal}}$ sends to $\mathcal{S}_{\mathsf{sh}}$ its randomness, inputs, auxiliary input, and the outputs received from T.

Outputs: Each uncorrupted party (i.e., not in $\mathcal{I}$) outputs whatever it received from T, the parties in $\mathcal{I}$ output nothing, and both $\mathcal{S}_{\mathsf{mal}}$ and $\mathcal{S}_{\mathsf{sh}}$ output some function of their respective view.

We next define the global view of the ideal world. For inputs $\mathbf{x} = (x_1, \ldots, x_n)$, security parameter $\kappa \in \mathbb{N}$, and adversaries $\mathcal{S}_{\mathsf{mal}}$ and $\mathcal{S}_{\mathsf{sh}}$ controlling the parties in $\mathcal{I} \subseteq \mathcal{P}$ and $\mathcal{H} \subseteq \mathcal{P} \setminus \mathcal{I}$, and holding auxiliary inputs z_{mal} and z_{sh}, respectively, we define the following. Let $\mathcal{S}_{\mathsf{FaF}} = (\mathcal{S}_{\mathsf{mal}}, \mathcal{S}_{\mathsf{sh}})$, let $z_{\mathsf{FaF}} = (z_{\mathsf{mal}}, z_{\mathsf{sh}})$, let $\text{MAL-VIEW}^{\mathsf{ideal},\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{mal}}}(\kappa, \mathbf{x}, z_{\mathsf{mal}})$ denote the output of $\mathcal{S}_{\mathsf{mal}}$, let $\text{SH-VIEW}^{\mathsf{ideal},\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{FaF}}}(\kappa, \mathbf{x}, z_{\mathsf{FaF}})$ denote the output of $\mathcal{S}_{\mathsf{sh}}$, and let $\text{OUT}^{\mathsf{ideal},\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{mal}}}(\kappa, \mathbf{x}, z_{\mathsf{mal}})$ denote the outputs of the uncorrupted parties (i.e., those in $\mathcal{P} \setminus \mathcal{I}$), in a random execution of the above ideal-world process when running alongside $\mathcal{S}_{\mathsf{mal}}$.

We denote the global view in the real model by

$$\text{IDEAL}^{\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{FaF}}}(\kappa, \mathbf{x}, z_{\mathsf{FaF}}) = \left(\text{MAL-VIEW}^{\mathsf{ideal},\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{mal}}}(\kappa, \mathbf{x}, z_{\mathsf{mal}}),\ \text{SH-VIEW}^{\mathsf{ideal},\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{FaF}}}(\kappa, \mathbf{x}, z_{\mathsf{FaF}}),\ \text{OUT}^{\mathsf{ideal},\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{mal}}}(\kappa, \mathbf{x}, z_{\mathsf{mal}})\right).$$

It will be convenient to denote

$$\text{MAL-IDEAL}^{\mathsf{mal},\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{mal}}}(\kappa, \mathbf{x}, z_{\mathsf{mal}}) = \left(\text{MAL-VIEW}^{\mathsf{ideal},\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{mal}}}(\kappa, \mathbf{x}, z_{\mathsf{mal}}),\ \text{OUT}^{\mathsf{ideal},\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{mal}}}(\kappa, \mathbf{x}, z_{\mathsf{mal}})\right),$$

i.e., the projection of $\text{IDEAL}^{\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{FaF}}}(\kappa, \mathbf{x}, z_{\mathsf{FaF}})$ to the output of $\mathcal{S}_{\mathsf{mal}}$ and the uncorrupted parties' output (those in $\mathcal{P} \setminus \mathcal{I}$), and denote

$$\text{SH-IDEAL}^{\mathsf{sh},\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{FaF}}}(\kappa, \mathbf{x}, z_{\mathsf{FaF}}) = \left(\text{SH-VIEW}^{\mathsf{ideal},\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{FaF}}}(\kappa, \mathbf{x}, z_{\mathsf{FaF}}),\ \text{OUT}^{\mathsf{ideal},\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{mal}}}(\kappa, \mathbf{x}, z_{\mathsf{mal}})\right),$$

i.e., the projection of $\text{IDEAL}^{\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{FaF}}}(\kappa, \mathbf{x}, z_{\mathsf{FaF}})$ to the output of $\mathcal{S}_{\mathsf{sh}}$ and the uncorrupted parties. When f is clear from the context, we will remove it for brevity.

Defining Strong FaF Security

Having defined the real and ideal models, we can now define the FaF security of protocols according to the real vs. ideal paradigm. We define only *strong* FaF security, which requires the simulatability of the joint view of both adversaries.

Definition 2.3 (Strong FaF security)**.** *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality, and let π be a protocol computing f. We say that π computes f with* computational (t,h)-strong FaF security*, if the following holds. For every malicious* PPT *adversary $\mathcal{A}_{\mathsf{mal}}$ controlling a set $\mathcal{I} \subseteq \mathcal{P}$ of size at most t in the real world, there exists a* PPT *adversary (called simulator) $\mathcal{S}_{\mathsf{mal}}$ controlling $\mathcal{I}$ in the ideal world; and for every subset of the remaining parties $\mathcal{H} \subseteq \mathcal{P} \setminus \mathcal{I}$ of size at most h controlled by a* semi-honest PPT *adversary $\mathcal{A}_{\mathsf{sh}}$, there exists a* PPT *adversary $\mathcal{S}_{\mathsf{sh}}$ controlling $\mathcal{H}$ in the ideal world, such that*

$$\left\{\text{IDEAL}^{\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{FaF}}}(\kappa, \mathbf{x}, z_{\mathsf{FaF}})\right\}_{\kappa \in \mathbb{N}, \mathbf{x} \in (\{0,1\}^*)^n, z_{\mathsf{FaF}} \in (\{0,1\}^*)^2} \overset{\text{C}}{\equiv} \left\{\text{REAL}^{\mathsf{FaF}}_{\pi,\mathcal{A}_{\mathsf{FaF}}}(\kappa, \mathbf{x}, z_{\mathsf{FaF}})\right\}_{\kappa \in \mathbb{N}, \mathbf{x} \in (\{0,1\}^*)^n, z_{\mathsf{FaF}} \in (\{0,1\}^*)^2}.$$

Statistical/perfect (t,h)-strong FaF security is defined similarly, with computational indistinguishability replaced with statistically close/identically distributed, and allowing the adversaries and simulators to be computationally unbounded.

2.1.1 Strong FaF Security-With-Abort

We next define strong FaF security-with-abort. In terms of definition, the only difference is how the ideal-world adversaries interact with the trusted party. We next describe this interaction.

The FaF Ideal World – Security-With-Abort

Unlike full security, here the malicious adversary can instruct the trusted party T to abort *after* receiving the output of the corrupted parties. Additionally, it is important to let the semi-honest simulator also receive the output of the semi-honest parties. This is because, in the real world, there is no guarantee that some of the honest parties will not learn the output. The ideal-world computation is parameterized by a (possibly randomized) n-party functionality $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ to compute. An ideal computation of f on input $\mathbf{x} = (x_1, \ldots, x_n)$ with security parameter κ in the presence of a malicious adversary (a simulator) $\mathcal{S}_{\mathsf{mal}}$ corrupting $\mathcal{I}$, and a semi-honest adversary $\mathcal{S}_{\mathsf{sh}}$ corrupting $\mathcal{H}$, proceeds as follows.

Inputs: Each party P_i holds 1^κ and a private input $x_i \in \{0,1\}^*$. The adversaries $\mathcal{S}_{\mathsf{mal}}$ and $\mathcal{S}_{\mathsf{sh}}$ are given auxiliary inputs $z_{\mathsf{mal}} \in \{0,1\}^*$ and $z_{\mathsf{sh}} \in \{0,1\}^*$, respectively, and the private input of every party controlled by them. The trusted party T holds 1^κ.

Parties send inputs: Each uncorrupted party $\mathsf{P}_i \in \mathcal{P} \setminus \mathcal{I}$ sends x_i as its input to T. For each corrupted party, the malicious adversary $\mathcal{S}_{\mathsf{mal}}$ sends to T a value from its domain. In case the adversary does not send any input, the trusted party replaces its input with a default value. Write $(x'_1, \ldots, x'_n)$ for the tuple of inputs received by the trusted party.

The trusted party performs computation: The trusted party T samples a random string rnd, computes $\mathbf{y} = (y_1, \ldots, y_n) = f(x'_1 \ldots, x'_n; \mathsf{rnd})$, and sends y_i to party P_i for every $i \in [n]$. If $\mathsf{P}_i \in \mathcal{I}$ then $\mathcal{S}_{\mathsf{mal}}$ receives y_i, and if $\mathsf{P}_i \in \mathcal{H}$ then $\mathcal{S}_{\mathsf{sh}}$ receives y_i.

The malicious adversary sends its (ideal-world) view: $\mathcal{S}_{\mathsf{mal}}$ sends to $\mathcal{S}_{\mathsf{sh}}$ its randomness, inputs, auxiliary input, and the outputs received from T.

The malicious adversary instructs the trusted party to continue or halt: The adversary $\mathcal{S}_{\mathsf{mal}}$ sends to T either $\mathsf{continue}$ or abort. If it sends $\mathsf{continue}$, then the trusted party sends y_j to P_j for every $j \in [n]$. Otherwise, if $\mathcal{S}_{\mathsf{mal}}$ sends abort, then T sends abort to all parties.

Outputs: Each uncorrupted party (i.e., not in $\mathcal{I}$) outputs whatever it received from T, the parties in $\mathcal{I}$ output nothing, and both $\mathcal{S}_{\mathsf{mal}}$ and $\mathcal{S}_{\mathsf{sh}}$ output some function of their respective view.

We next define the global view of the ideal world. For inputs $\mathbf{x} = (x_1, \ldots, x_n)$, security parameter $\kappa \in \mathbb{N}$, and adversaries $\mathcal{S}_{\mathsf{mal}}$ and $\mathcal{S}_{\mathsf{sh}}$ controlling the parties in $\mathcal{I} \subseteq \mathcal{P}$ and $\mathcal{H} \subseteq \mathcal{P} \setminus \mathcal{I}$, and holding auxiliary inputs z_{mal} and z_{sh}, respectively, we define the following. Let $\mathcal{S}_{\mathsf{FaF}} = (\mathcal{S}_{\mathsf{mal}}, \mathcal{S}_{\mathsf{sh}})$, let $z_{\mathsf{FaF}} = (z_{\mathsf{mal}}, z_{\mathsf{sh}})$, and let $\text{IDEAL}^{\mathsf{swa},\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{FaF}}}(\kappa, \mathbf{x}, z_{\mathsf{FaF}})$ denote the joint outputs of $\mathcal{S}_{\mathsf{mal}}$, $\mathcal{S}_{\mathsf{sh}}$, and the uncorrupted parties (i.e., those in $\mathcal{P} \setminus \mathcal{I}$), in a random execution of the above ideal-world process when running alongside $\mathcal{S}_{\mathsf{mal}}$.

Having described the computation of the ideal world for security-with-abort, we are now ready to define dynamic security-with-abort.

Definition 2.4 (Strong FaF security-with-abort)**.** *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality, and let π be a protocol computing f. We say that π computes f with* computational (t,h)-strong FaF security-with-abort*, if the following holds. For every malicious* PPT *adversary $\mathcal{A}_{\mathsf{mal}}$ controlling a set $\mathcal{I} \subseteq \mathcal{P}$ of size at most t in the real world, there exists a* PPT *adversary (called simulator) $\mathcal{S}_{\mathsf{mal}}$ controlling $\mathcal{I}$ in the ideal world; and for every subset of the remaining parties $\mathcal{H} \subseteq \mathcal{P} \setminus \mathcal{I}$ of size at most h controlled by a* semi-honest PPT *adversary $\mathcal{A}_{\mathsf{sh}}$, there exists a* PPT *adversary $\mathcal{S}_{\mathsf{sh}}$ controlling $\mathcal{H}$ in the ideal world, such that*

$$\left\{\mathrm{IDEAL}^{\mathsf{swa},\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{FaF}}}(\kappa,\mathbf{x},z_{\mathsf{FaF}})\right\}_{\kappa\in\mathbb{N},\mathbf{x}\in(\{0,1\}^*)^n,z_{\mathsf{FaF}}\in(\{0,1\}^*)^2} \stackrel{\mathrm{C}}{\equiv} \left\{\mathrm{REAL}^{\mathsf{FaF}}_{\pi,\mathcal{A}_{\mathsf{FaF}}}(\kappa,\mathbf{x},z_{\mathsf{FaF}})\right\}_{\kappa\in\mathbb{N},\mathbf{x}\in(\{0,1\}^*)^n,z_{\mathsf{FaF}}\in(\{0,1\}^*)^2}.$$

Statistical/perfect (t,h)-strong FaF security-with-abort is defined similarly, with computational indistinguishability replaced with statistically close/identically distributed, and allowing the adversaries and simulators to be computationally unbounded.

2.2 Security Model: Adaptive Security

In this section, we define adaptive security, where the adversary is allowed to corrupt new parties during the protocol's execution. Whenever the adversary corrupts a new party, it learns the party's internal state, which includes the party's input, output, randomness, incoming messages, and also the state from previous runs. To capture this, the model includes an additional Turing machine $\mathcal{Z}$ called the *environment*, whose role is to provide the adversary with auxiliary information about the newly corrupted parties.

The Real World

An n-party protocol π is defined by a set of n PPT interactive Turing machines $\mathcal{P} = \{\mathsf{P}_1,\ldots,\mathsf{P}_n\}$. Each Turing machine (party) P_i holds at the beginning of the execution the common security parameter 1^κ, a private input x_i, and random coins. The adversary $\mathcal{A}$ is another interactive Turing machine describing the behavior of the corrupted parties. It starts the execution with some randomness. The environment $\mathcal{Z}$ is an interactive Turing machine that starts with an auxiliary input z and some randomness.

The execution starts with the environment sending a message $z_{\mathcal{A}}$ to $\mathcal{A}$ (this corresponds to auxiliary input static corruptions). The execution then proceeds in synchronous rounds, where each round ρ consists of mini-rounds described as follows. The adversary $\mathcal{A}$ first chooses which parties to corrupt. Upon corrupting a party P_i, $\mathcal{A}$ receives the party's input, randomness, and all messages it received in previous rounds. Additionally, $\mathcal{A}$ sends "corrupt P_i" to $\mathcal{Z}$, who responds with some additional auxiliary information z_i. It then chooses an uncorrupted party P_i that has not been activated this round and activates it. Upon activation, P_i receives the messages sent to it in the previous round, generates its messages for this round, and the next mini-round begins. The adversary learns the messages sent by P_i to all (currently) corrupted parties. Once all the uncorrupted parties were activated, the adversary generates the messages to be sent by the corrupted parties that were not yet activated in this round, and the next round begins (note that this allows the adversary to be rushing).

Throughout the execution of the protocol, all uncorrupted parties follow the instructions of the prescribed protocol, whereas the corrupted parties receive their instructions from the adversary. A

semi-honest adversary instructs the corrupted parties to follow the protocol. A *malicious* adversary can instruct the corrupted parties to deviate from the protocol in any arbitrary way. At the conclusion of the protocol, the honest parties output a value as specified by the protocol, the corrupted parties output nothing, and the adversary outputs some function of its view (which includes its randomness, the auxiliary information received from $\mathcal{Z}$, and each corrupted party's input, randomness, and all messages it received).

Finally, the "post-execution corruption" phase begins. First, the environment receives the output of the parties and the adversary. Then, $\mathcal{Z}$ and $\mathcal{A}$ interact in several rounds. Each round $\mathcal{Z}$ sends "corrupt P_i" to $\mathcal{A}$ for some $i \in [n]$. Upon receiving this message, $\mathcal{A}$ responds with some message (intuitively, this message is interpreted as P_i's internal data). This interaction continues until $\mathcal{Z}$ halts with some output.

We say that the adversary is (t, h)-*limited* if it corrupts at most t parties during the execution of the protocol (excluding the post-execution phase), and corrupts at most h parties during the post-execution phase.[7] We stress that if $\mathcal{A}$ is (t, h)-limited and already corrupted h parties during the post-execution phase, then upon receiving "corrupt P_i" from $\mathcal{Z}$ it ignores this message.

For inputs $\mathbf{x} = (x_1, \ldots, x_n)$, security parameter κ, adversary $\mathcal{A}$, and environment $\mathcal{Z}$ holding auxiliary input z, we let $\text{VIEW}^{\mathsf{real},\mathsf{adp}}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, \mathbf{x}, z)$ denote the output of $\mathcal{A}$, let $\text{OUT}^{\mathsf{real},\mathsf{adp}}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, \mathbf{x}, z)$ denote the output of the honest parties, and we let $\text{REAL}^{\mathsf{adp}}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, \mathbf{x}, z)$ denote the output of $\mathcal{Z}$ in a random execution of the real-world protocol π.

The Ideal World

We consider an ideal computation with *guaranteed output delivery* (also referred to as full security), where a trusted party T performs the computation on behalf of the parties, and the ideal-model adversary cannot abort the computation. The ideal-world computation is parameterized by a (possibly randomized) n-party functionality $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ to compute. An ideal computation of f on input $\mathbf{x} = (x_1, \ldots, x_n)$ with security parameter κ in the presence of an adversary (a simulator) $\mathcal{S}$, and environment $\mathcal{Z}$, proceeds as follows.

First corruption stage: The process starts with $\mathcal{Z}$ computing and sending some auxiliary information $z_{\mathcal{S}}$ to $\mathcal{S}$. Next, $\mathcal{S}$ and $\mathcal{Z}$ interact in several rounds, where in each round the adversary $\mathcal{S}$ may corrupt a party P_i. In such a scenario, the adversary receives the party's input and sends "corrupt P_i" to $\mathcal{Z}$. The environment computes and sends additional auxiliary information z_i to $\mathcal{S}$.

The parties send inputs to the trusted party: Each honest party sends its input to the trusted party T. For each corrupted party, the adversary $\mathcal{S}$ sends to T a value from its domain as its input. In case the adversary does not send any input, the trusted party replaces its input with a default value. Let $\mathbf{x}'$ denote the vector of inputs received by T.

The trusted party performs the computation: The trusted party samples randomness rnd, computes $(y_1, \ldots, y_n) = f(\mathbf{x}; \mathsf{rnd})$, and sends y_i to P_i for every $i \in [n]$. If P_i is corrupted then $\mathcal{S}$ receives y_i as well.

[7] The standard definition considers a t-limited adversary that can corrupt at most t parties during both the execution of the protocol and the post-execution corruption phase. For the comparison with our new definition (see Section 3), it will be instructive to separate corruption during the protocol, and after it terminated.

Second corruption stage: After learning the output, $\mathcal{S}$ and $\mathcal{Z}$ interact in another sequence of iterations, similar to the first corruption stage. The only difference is that for any new corrupted party, $\mathcal{S}$ also receives the output of the party it corrupts.

Output: The honest parties output whatever they received from the trusted party, the corrupted parties output nothing, and the adversary outputs some function of its view.

Post-execution corruption: The environment receives the output of $\mathcal{S}$ and the output of the honest parties. Then, $\mathcal{Z}$ and $\mathcal{S}$ interact in several rounds. In each round $\mathcal{Z}$ sends "corrupt P_i" to $\mathcal{S}$ for some $i \in [n]$. Upon receiving this message, $\mathcal{S}$ responds with some message (intuitively, this message is interpreted as P_i's internal data). This interaction continues until $\mathcal{Z}$ halts with some output.

Similarly to the real world, we say that the adversary is (t,h)-limited if it corrupts at most t parties before the post-execution phase, and corrupts at most h parties during the post-execution phase. If $\mathcal{S}$ is (t,h)-limited and already corrupted h parties during the post-execution phase, then upon receiving "corrupt P_i" from $\mathcal{Z}$ it ignores this message.

For inputs $\mathbf{x} = (x_1, \ldots, x_n)$, security parameter κ, adversary $\mathcal{S}$, and an environment $\mathcal{Z}$ holding auxiliary input z, we let $\text{VIEW}^{\mathsf{ideal},\mathsf{adp}}_{f,\mathcal{S},\mathcal{Z}}(\kappa,\mathbf{x},z)$ denote the output of $\mathcal{S}$, we let $\text{OUT}^{\mathsf{ideal},\mathsf{adp}}_{f,\mathcal{S},\mathcal{Z}}(\kappa,\mathbf{x},z)$ denote the output of the honest parties, and let $\text{IDEAL}^{\mathsf{adp}}_{f,\mathcal{S},\mathcal{Z}}(\kappa,\mathbf{x},z)$ denote the output of $\mathcal{Z}$, in a random execution of the above ideal world.

Defining Adaptive Security

Having described the computation in both the real and ideal worlds, we are now ready to define adaptive security.

Definition 2.5 (Adaptive security)**.** *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality, and let π be a protocol computing f. We say that π computes f with* computational (t,h)-adaptive security *if for every (t,h)-limited* PPT *adversary $\mathcal{A}$ and every* PPT *environment $\mathcal{Z}$, there exists a (t,h)-limited* PPT *ideal-world adversary (called simulator) $\mathcal{S}$ such that*

$$\left\{\text{REAL}^{\mathsf{adp}}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa,\mathbf{x},z)\right\}_{\kappa\in\mathbb{N},\mathbf{x}\in(\{0,1\}^*)^n,z\in\{0,1\}^*} \overset{\text{C}}{\equiv} \left\{\text{IDEAL}^{\mathsf{adp}}_{f,\mathcal{S},\mathcal{Z}}(\kappa,\mathbf{x},z)\right\}_{\kappa\in\mathbb{N},\mathbf{x}\in(\{0,1\}^*)^n,z\in\{0,1\}^*}.$$

Statistical/perfect (t,h)-adaptive security are defined similarly, with computational indistinguishability replaced with statistically close/identically distributed, and allowing the adversary, the simulator, and the environment to be computationally unbounded.

3 A New Security Definition: Dynamic Security

We next introduce our new security definition for dynamic adversaries. The definition captures security against adversaries that corrupt parties during the execution of the protocol, however, unlike adaptive security, the adversary cannot choose who to corrupt and when. This is modeled similarly to adaptive security, where there is an environment interacting with the adversary. The main difference is that *before* the start of the protocol, the environment chooses which parties the adversary corrupts and at what round it will corrupt them. In particular, this means that the choice of corruption is independent of the execution of the protocol. To differentiate from the adaptive

security definition, we will refer to the environment as a *dynamic* environment. We next provide a formal treatment and provide details on the interaction in the real and ideal worlds, followed by the security definition.

The Real World – Interaction With a Dynamic Adversary

An n-party protocol π is defined by a set of n PPT interactive Turing machines $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$. Each Turing machine (party) P_i holds at the beginning of the execution the common security parameter 1^κ, a private input x_i, and random coins. The adversary $\mathcal{A}$ is another interactive Turing machine describing the behavior of the corrupted parties. It starts the execution with some randomness. The dynamic environment $\mathcal{Z}$ is an interactive Turing machine that starts with an auxiliary input z and some randomness.

The execution starts with the dynamic environment generating $(\mathcal{L}, z_\mathcal{A}, (z_i)_{i\in[n]}) \leftarrow \mathcal{Z}(z)$ and sending $z_\mathcal{A}$ to $\mathcal{A}$ (this corresponds to the auxiliary information of $\mathcal{A}$ in the static case). Here, $\mathcal{L}$ is a *corruption list* $\mathcal{L} = (\rho_i)_{i\in[n]} \in ([r] \cup \{\mathsf{PEC}, \mathsf{honest}\})^n$, where r is the number of rounds in π. The corruption list indicates which parties $\mathcal{A}$ is going to corrupt and when. Roughly, if $\rho_i \in [r]$ then $\mathcal{A}$ corrupts P_i at round ρ_i, if $\rho_i = \mathsf{PEC}$ then $\mathcal{A}$ corrupts P_i after the execution of the protocol terminated (see the post-execution corruption phase below), and if $\rho_i = \mathsf{honest}$ then $\mathcal{A}$ never corrupts P_i. We call the environment (t, h)-limited if

$$|\{i \in [n] : \rho_i \in [r]\}| \leq t \quad \text{and} \quad |\{i \in [n] : \rho_i = \mathsf{PEC}\}| \leq h.$$

The execution then proceeds in synchronous rounds, where each round ρ consists of mini-rounds described as follows. In the first mini-round, for every $i \in [n]$ such that $\rho_i = \rho$, the environment sends "corrupt P_i" to the adversary. The adversary then corrupts party P_i and receives the party's input, randomness, and all messages sent and received by the party. Additionally, the environment sends z_i to $\mathcal{A}$ as additional auxiliary information. Next, the adversary $\mathcal{A}$ chooses an uncorrupted party P_i that has not been activated this round and activates it. The activated party receives all messages sent to it in the previous round, generate its new messages for this round, and the next mini-round begins. The adversary receives all messages sent by P_i to the currently corrupted parties. Once all uncorrupted parties were activated, the adversary generates messages to be sent by the corrupted parties that were not activated this round, and the next round begins (note that this allows the adversary to be rushing).

Throughout the execution of the protocol, all uncorrupted parties follow the instructions of the prescribed protocol, whereas the corrupted parties receive their instructions from the adversary. A *semi-honest* adversary instructs the corrupted party to follow the protocol. A *malicious* adversary can instruct the corrupted parties to deviate from the protocol in any arbitrary way. At the conclusion of the protocol, the honest parties output a value as specified by the protocol, the corrupted parties output nothing, and the adversary outputs some function of its view (which includes its randomness, the auxiliary information received from $\mathcal{Z}$, and each corrupted party's input, randomness, and all messages it received).

Finally, the "post-execution corruption" phase begins. The environment first receives the output of $\mathcal{A}$ and the output of the honest parties. It then sends to $\mathcal{A}$ "corrupt P_i" for every $i \in [n]$ such that $\rho_i = \mathsf{PEC}$. For every such party, the adversary receives its input, randomness, and all messages it sent and received during the execution of the protocol. Additionally, it receives from $\mathcal{Z}$ the additional auxiliary information z_i. The adversary then sends to $\mathcal{Z}$ some message. Finally, the environment generates some output and halts.

For inputs $\mathbf{x} = (x_1, \ldots, x_n)$, security parameter κ, adversary $\mathcal{A}$, and environment $\mathcal{Z}$ holding auxiliary input z, we let $\text{VIEW}^{\mathsf{real},\mathsf{dyn}}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, \mathbf{x}, z)$ denote the output of $\mathcal{A}$, let $\text{OUT}^{\mathsf{real},\mathsf{dyn}}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, \mathbf{x}, z)$ denote the output of the honest parties, and we let $\text{REAL}^{\mathsf{dyn}}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, \mathbf{x}, z)$ denote the output of $\mathcal{Z}$ in a random execution of the real-world protocol π.

The Ideal World – Interaction With a Dynamic Adversary

We consider an ideal computation with *guaranteed output delivery* (also referred to as full security), where a trusted party T performs the computation on behalf of the parties, and the ideal-model adversary cannot abort the computation. The ideal-world computation is parameterized by a (possibly randomized) n-party functionality $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ to compute. An ideal computation of f on input $\mathbf{x} = (x_1, \ldots, x_n)$ with security parameter κ in the presence of an adversary (a simulator) $\mathcal{S}$, and a dynamic environment $\mathcal{Z}$, proceeds as follows.

First corruption stage: The process starts with $\mathcal{Z}$ generating $(\mathcal{L}, z_{\mathcal{S}}, (z_i)_{i\in[n]})$, where $\mathcal{L}$ is a corruption list $\mathcal{L} = (\rho_i)_{i\in[n]} \in ([r] \cup \{\mathsf{PEC}, \mathsf{honest}\})^n$. It then sends $z_{\mathcal{S}}$ to $\mathcal{S}$. Next, $\mathcal{S}$ and $\mathcal{Z}$ interact in iteration, where in each iteration the environment may instruct $\mathcal{S}$ to corrupt a party P_i such that $\rho_i \in [r]$. In such a scenario, the adversary receives the party's input, and $\mathcal{Z}$ sends z_i to $\mathcal{S}$.

The parties send their inputs to the trusted party: Each honest party sends its input to the trusted party T. For each corrupted party, the adversary $\mathcal{S}$ sends to T a value from its domain as its input. In case the adversary does not send any input, the trusted party replaces its input with a default value. Let $\mathbf{x}' = (x'_1, \ldots, x'_n)$ denote the vector of inputs received by T.

The trusted party performs the computation: The trusted party samples randomness rnd, computes $(y_1, \ldots, y_n) = f(\mathbf{x}; \mathsf{rnd})$, and sends y_i to P_i for every $i \in [n]$. If P_i is corrupted then $\mathcal{S}$ receives y_i as well.

Second corruption stage: After learning the output, $\mathcal{S}$ and $\mathcal{Z}$ interact in another sequence of iterations, similar to the first corruption stage. The only difference is that for any new corrupted party, $\mathcal{S}$ also receives the output of the party it corrupts.

Output: The honest parties output whatever they received from the trusted party, the corrupted parties output nothing, and the adversary outputs some function of its view.

Post-execution corruption: The environment receives the output of $\mathcal{S}$ and the output of the honest parties. It then sends "corrupt P_i" for every $i \in [n]$ such that $\rho_i = \mathsf{PEC}$. For every such party, the adversary receives its input and output, and some additional auxiliary information z_i from $\mathcal{Z}$. The adversary then sends to $\mathcal{Z}$ some message. Finally, the environment generates some output and halts.

For inputs $\mathbf{x} = (x_1, \ldots, x_n)$, security parameter κ, adversary $\mathcal{S}$, and an environment $\mathcal{Z}$ holding auxiliary input z, we let $\text{VIEW}^{\mathsf{ideal},\mathsf{dyn}}_{f,\mathcal{S},\mathcal{Z}}(\kappa, \mathbf{x}, z)$ denote the output of $\mathcal{S}$, we let $\text{OUT}^{\mathsf{ideal},\mathsf{dyn}}_{f,\mathcal{S},\mathcal{Z}}(\kappa, \mathbf{x}, z)$ denote the output of the honest parties, and let $\text{IDEAL}^{\mathsf{dyn}}_{f,\mathcal{S},\mathcal{Z}}(\kappa, \mathbf{x}, z)$ denote the output of $\mathcal{Z}$, in a random execution of the above ideal world.

Defining Dynamic Security

Having described the computation in both the real and ideal worlds, we are now ready to define dynamic security.

Definition 3.1 (Dynamic security)**.** *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality, and let π be a protocol computing f. We say that π computes f with* computational (t,h)-dynamic security *if the following holds. For every* PPT *adversary $\mathcal{A}$ and every* PPT *(t,h)-limited dynamic environment $\mathcal{Z}$, there exists a* PPT *ideal-world adversary (called simulator) $\mathcal{S}$ such that*

$$\left\{\mathrm{REAL}^{\mathsf{dyn}}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa,\mathbf{x},z)\right\}_{\kappa\in\mathbb{N},\mathbf{x}\in(\{0,1\}^*)^n,z\in\{0,1\}^*} \overset{\mathrm{C}}{\equiv} \left\{\mathrm{IDEAL}^{\mathsf{dyn}}_{f,\mathcal{S},\mathcal{Z}}(\kappa,\mathbf{x},z)\right\}_{\kappa\in\mathbb{N},\mathbf{x}\in(\{0,1\}^*)^n,z\in\{0,1\}^*}.$$

Statistical/perfect (t,h)-dynamic security are defined similarly, with computational indistinguishability replaced with statistically close/identically distributed, and allowing the adversary, the simulator, and the environment to be computationally unbounded.

3.1 Dynamic Security-With-Abort

In this section, we define the secure-with-abort variant of dynamic security. In terms of definition, the only difference is how the ideal-world adversary interacts with the trusted party. We next describe this interaction.

The Ideal World – Dynamic Security-With-Abort

Unlike full security, here the adversary can instruct the trusted party T to abort *after* the second corruption stage, and in particular, after receiving the output of the (currently) corrupted parties. This is because in the real world, there is no guarantee that some of the honest parties do not learn the output. The ideal-world computation is parameterized by a (possibly randomized) n-party functionality $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ to compute. On input $\mathbf{x} = (x_1,\ldots,x_n)$ with security parameter κ in the presence of an adversary (a simulator) $\mathcal{S}$, and a dynamic environment $\mathcal{Z}$, the ideal-world computation proceeds as follows.

First corruption stage: The process starts with $\mathcal{Z}$ generating $(\mathcal{L}, z_{\mathcal{S}}, (z_i)_{i\in[n]})$, where $\mathcal{L}$ is a corruption list $\mathcal{L} = (\rho_i)_{i\in[n]} \in ([r] \cup \{\mathsf{PEC}, \mathsf{honest}\})^n$. It then sends $z_{\mathcal{S}}$ to $\mathcal{S}$. Next, $\mathcal{S}$ and $\mathcal{Z}$ interact in iteration, where in each iteration the environment may instruct $\mathcal{S}$ to corrupt a party P_i such that $\rho_i \in [r]$. In such a scenario, the adversary receives the party's input, and $\mathcal{Z}$ sends z_i to $\mathcal{S}$. Additionally, the trusted party learns the identity of the newly corrupted party. Let $\mathcal{I} \subseteq [n]$ denote the set of parties corrupted during this stage.

The parties send their inputs to the trusted party: Each honest party sends its input to the trusted party T. For each corrupted party, the adversary $\mathcal{S}$ sends to T a value from its domain as its input. In case the adversary does not send any input, the trusted party replaces its input with a default value. Let $\mathbf{x}' = (x'_1,\ldots,x'_n)$ denote the vector of inputs received by T.

The trusted party performs the computation: The trusted party samples randomness rnd, computes $(y_1,\ldots,y_n) = f(\mathbf{x};\mathsf{rnd})$, and sends $(y_i)_{i\in\mathcal{I}}$ to $\mathcal{S}$.

Second corruption stage: After learning the output, $\mathcal{S}$ and $\mathcal{Z}$ interact in another sequence of iterations, similar to the first corruption stage. The only difference is that for any new corrupted party, $\mathcal{S}$ also receives the output of the party it corrupts (sent to it by the trusted party).

The adversary instructs the trusted party to continue or halt: The adversary $\mathcal{S}$ sends to T either continue or abort. If it sends continue, then the trusted party sends y_i to P_i for every $i \in [n]$. Otherwise, if $\mathcal{S}$ sends abort, then T sends abort to all parties.

Output: The honest parties output whatever they received from the trusted party, the corrupted parties output nothing, and the adversary outputs some function of its view.

Post-execution corruption: The environment receives the output of $\mathcal{S}$ and the output of the honest parties. It then sends "corrupt P_i" for every $i \in [n]$ such that $\rho_i = \mathsf{PEC}$. For every such party, the adversary receives its input and output, and some additional auxiliary information z_i from $\mathcal{Z}$. The adversary then sends to $\mathcal{Z}$ some message. Finally, the environment generates some output and halts.

Defining Dynamic Security-With-Abort

Having described the computation of the ideal world for security-with-abort, we are now ready to define dynamic security-with-abort.

Definition 3.2 (Dynamic security)**.** *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality, and let π be a protocol computing f. We say that π computes f with* computational (t,h)-dynamic security-with-abort *if the following holds. For every* PPT *adversary $\mathcal{A}$ and every* PPT *(t,h)-limited dynamic environment $\mathcal{Z}$, there exists a* PPT *ideal-world adversary (called simulator) $\mathcal{S}$ such that*

$$\left\{\mathrm{REAL}^{\mathsf{dyn}}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa,\mathbf{x},z)\right\}_{\kappa\in\mathbb{N},\mathbf{x}\in(\{0,1\}^*)^n,z\in\{0,1\}^*} \stackrel{\mathrm{C}}{\equiv} \left\{\mathrm{IDEAL}^{\mathsf{swa},\mathsf{dyn}}_{f,\mathcal{S},\mathcal{Z}}(\kappa,\mathbf{x},z)\right\}_{\kappa\in\mathbb{N},\mathbf{x}\in(\{0,1\}^*)^n,z\in\{0,1\}^*}.$$

Statistical/perfect (t,h)-dynamic security-with-abort are defined similarly, with computational indistinguishability replaced with statistically close/identically distributed, and allowing the adversary, the simulator, and the environment to be computationally unbounded.

4 Equivalence of Dynamic Security and Strong FaF Security

We show that strong FaF security is equivalent to dynamic security. This holds for both full security and security-with-abort.

Theorem 4.1. *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality, let π be a protocol computing f, and* $\mathsf{type} \in \{$*computational*, *statistical*, *perfect*$\}$*, and let* $\mathsf{security} \in \{$*security*, *security-with-abort*$\}$*. Then π computes f with* type *(t,h)-dynamic* security *if and only if π computes f with* type *(t,h)-strong FaF* security*.*

Proof. We prove the result for type = computational and security = security. The other cases are handled similarly.

For the first direction, assume that π computes f with computational (t,h)-dynamic security. Fix a malicious adversary $\mathcal{A}_{\mathsf{mal}}$ that holds auxiliary input z_{mal} and corrupts the parties in $\mathcal{I} \subseteq \mathcal{P}$,

and fix a semi-honest adversary $\mathcal{A}_{\mathsf{sh}}$ that holds auxiliary input z_{sh} and corrupts the parties in $\mathcal{H} \subseteq \mathcal{P} \setminus \mathcal{I}$. Let $\mathcal{L} = (\rho_i)_{i\in[n]}$ be the corruption list defined as follows. For every $i \in [n]$ such that $\mathsf{P}_i \in \mathcal{I}$ let $\rho_i = 1$, for every $i \in [n]$ such that $\mathsf{P}_i \in \mathcal{H}$ let $\rho_i = \mathsf{PEC}$, and for any remaining $i \in [n]$ let $\rho_i = \mathsf{honest}$. Additionally, for every $i \in [n]$ let $z_i = z_{\mathsf{sh}}$ if $\mathsf{P}_i \in \mathcal{H}$, and let $z_i = \lambda$ otherwise. Consider a (t, h)-limited dynamic environment $\mathcal{Z}$ that is given the auxiliary input $z = (z_{\mathsf{mal}}, z_{\mathsf{sh}})$, and outputs $(\mathcal{L}, z_{\mathsf{mal}}, (z_i)_{i\in[n]})$. Additionally, in the post-execution phase, it outputs all the information it received (i.e., the output of $\mathcal{A}$ during the protocol's execution, the message $\mathcal{A}$ sent to it in the post-execution phase, and the output of the parties in $\mathcal{P} \setminus \mathcal{I}$).

Consider a dynamic adversary $\mathcal{A}_{\mathsf{dyn}}$ that during the protocol behaves the same as $\mathcal{A}_{\mathsf{mal}}$ does, and after the protocol terminates it runs $\mathcal{A}_{\mathsf{sh}}$ and sends the output to $\mathcal{Z}$. Then by security assumption, there exists a simulator $\mathcal{S}_{\mathsf{dyn}}$ for $\mathcal{A}_{\mathsf{dyn}}$. Define the simulator $\mathcal{S}_{\mathsf{mal}}$ for $\mathcal{A}_{\mathsf{mal}}$, controlling the parties in $\mathcal{I}$, as follows. Run $\mathcal{S}_{\mathsf{dyn}}$ while taking the role of the environment (i.e., by sending "corrupt P_i" and the needed information when necessary). When $\mathcal{S}_{\mathsf{dyn}}$ sends inputs to the trusted party for the set of currently corrupted parties $\mathcal{I}' \subseteq \mathcal{I}$, send to T the same inputs as $\mathcal{S}_{\mathsf{dyn}}$ used, and for every $\mathsf{P}_i \in \mathcal{I} \setminus \mathcal{I}'$ send its input x_i to T. Continue running $\mathcal{S}_{\mathsf{dyn}}$ as before and output whatever it outputs. Clearly for all inputs $\mathbf{x} \in (\{0,1\}^*)^n$ and every auxiliary input $z \in \{0,1\}^*$,

$$\text{MAL-IDEAL}^{\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{mal}}}(\kappa, \mathbf{x}, z_{\mathsf{mal}}) \equiv \left(\text{MAL-VIEW}^{\mathsf{ideal},\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{dyn}},\mathcal{Z}}(\kappa, \mathbf{x}, z),\ \text{OUT}^{\mathsf{ideal},\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{dyn}},\mathcal{Z}}(\kappa, \mathbf{x}, z)\right).$$

Next, we define the semi-honest simulator $\mathcal{S}_{\mathsf{sh}}$ for $\mathcal{A}_{\mathsf{sh}}$ to output whatever $\mathcal{S}_{\mathsf{dyn}}$ sent to $\mathcal{Z}$ during the post-execution phase. Note that since $\mathcal{S}_{\mathsf{sh}}$ receives the ideal-world view of $\mathcal{S}_{\mathsf{mal}}$, the output of $\mathcal{S}_{\mathsf{sh}}$ conditioned on the output of $\mathcal{S}_{\mathsf{mal}}$ and the uncorrupted parties is identically distributed to the message $\mathcal{S}_{\mathsf{dyn}}$ sent to $\mathcal{Z}$ in the post-execution phase conditioned on its output and the honest parties' output. Thus,

$$\text{IDEAL}^{\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{mal}},\mathcal{S}_{\mathsf{sh}}}(\kappa, \mathbf{x}, z_{\mathsf{mal}}, z_{\mathsf{sh}}) \equiv \text{IDEAL}^{\mathsf{dyn}}_{f,\mathcal{S}_{\mathsf{dyn}},\mathcal{Z}}(\kappa, \mathbf{x}, z).$$

As the same holds in the real world, we conclude they are computationally indistinguishable.

We now prove the second direction. We first introduce the notion of a *partial corruption list*. Given a corruption list $\mathcal{L} = (\rho_i)_{i\in[n]}$ we define its partial corruption list, denoted $\mathcal{L}^* = (\rho^*_i)_{i\in[n]} \in ([r] \cup \{*\})^n$, as follows. For every $i \in [n]$ such that $\rho_i \in [r]$ let $\rho^*_i = \rho_i$, and $\rho^*_i = *$ for every other $i \in [n]$.

Assume that π computes f with computational (t, h)-strong FaF security, and let r denote its number of rounds. Fix an adversary $\mathcal{A}_{\mathsf{dyn}}$ and a (t, h)-limited dynamic environment $\mathcal{Z}$ with auxiliary input z. For every partial corruption list $\mathcal{L}^* = (\rho^*_i)_{i\in[n]} \in ([r] \cup \{*\})^n$ in the support of $\mathcal{Z}$, define the malicious adversary $\mathcal{A}_{\mathcal{L}^*}$ as follows. It starts with auxiliary input $z := (z_{\mathcal{A}}, (z_i)_{i:\rho^*_i\in[r]})$ (i.e., it consists of all the auxiliary information that $\mathcal{Z}$ sends to $\mathcal{A}$ during the protocol), corrupts the parties in $\mathcal{I} := \{\mathsf{P}_i : \rho^*_i \in [r]\}$, and does the following in every round $\rho \in [r]$.

1. Send to $\mathcal{A}_{\mathsf{dyn}}$ "corrupt P_i" for every $i \in [n]$ such that $\rho^*_i = \rho$. Let $\mathcal{I}_\rho$ denote the set of all parties that $\mathcal{A}_{\mathsf{dyn}}$ controls at round ρ.

2. For every $i \in [n]$ such that $\mathsf{P}_i \in \mathcal{I} \setminus \mathcal{I}_\rho$ (i.e., every party that has yet to be corrupted by $\mathcal{A}_{\mathsf{dyn}}$), compute the messages that an honest P_i computes as specified by the protocol and send them to the honest parties.

3. Send to $\mathcal{A}_{\mathsf{dyn}}$ the messages received from the honest parties, as well the messages computed for the parties in $\mathcal{I} \setminus \mathcal{I}_\rho$.

4. $\mathcal{A}_{\mathsf{dyn}}$ then responds with messages for the parties in $\mathcal{I}_\rho$. Send those messages to the honest parties.

After the protocol terminated, we let $\mathcal{A}_{\mathcal{L}^*}$ broadcast its view. Next, for every corruption list $\mathcal{L} = (\rho_i)_{i\in[n]}$ define the following semi-honest adversary $\mathcal{A}_{\mathsf{sh},\mathcal{L}}$. It is given the auxiliary input $z_{\mathsf{sh}} = (z_i)_{i:\rho_i=\mathsf{PEC}}$, it controls the parties in $\mathcal{H} := \{\mathsf{P}_i : \rho_i = \mathsf{PEC}\}$, and after receiving the broadcast message of the malicious adversary, send it to $\mathcal{A}_{\mathsf{dyn}}$ and output whatever it sends to $\mathcal{Z}$ at the post-execution phase.

By the security assumption, for every corruption list $\mathcal{L}$ there exist simulators $\mathcal{S}_{\mathcal{L}^*}$ and $\mathcal{S}_{\mathsf{sh},\mathcal{L}}$ for $\mathcal{A}_{\mathcal{L}^*}$ and $\mathcal{A}_{\mathsf{sh},\mathcal{L}}$, respectively, where $\mathcal{L}^*$ is the partial list that corresponds to $\mathcal{L}$. We are now ready to define the simulator $\mathcal{S}_{\mathsf{dyn}}$ for $\mathcal{A}_{\mathsf{dyn}}$.

1. In the first corruption stage, interact with $\mathcal{Z}$ for r rounds. At every round $\rho \in [r]$, $\mathcal{Z}$ may instruct the simulator to corrupt a party P_i.

2. Define the partial corruption list $\mathcal{L}^* = (\rho_i)_{i\in[n]}$ as follows. For every $i \in [n]$ let $\rho_i = \rho$ if $\mathcal{Z}$ sent "corrupt P_i" at round ρ, and let $\rho_i = *$ otherwise.

3. Run $\mathcal{S}_{\mathcal{L}^*}$ and output whatever it outputs (sending the same input to the trusted party).

4. To simulate the post-execution phase, first receive "corrupt P_i" for every $i \in [n]$ such that $\rho_i = \mathsf{PEC}$ from $\mathcal{Z}$. This completes $\mathcal{L}^*$ to a corruption list $\mathcal{L}$. Run the semi-honest simulator $\mathcal{S}_{\mathsf{sh},\mathcal{L}}$ and send to $\mathcal{Z}$ whatever $\mathcal{S}_{\mathsf{sh},\mathcal{L}}$ outputs.

Now, observe that in the real world, for all inputs $\mathbf{x} \in (\{0,1\}^*)^n$ and every auxiliary input $z \in \{0,1\}^*$,

$$\begin{aligned}&\left(\mathrm{VIEW}^{\mathsf{real},\mathsf{dyn}}_{\pi,\mathcal{A}_{\mathsf{dyn}},\mathcal{Z}}(\kappa,\mathbf{x},z),\ \mathrm{OUT}^{\mathsf{real},\mathsf{dyn}}_{\pi,\mathcal{A}_{\mathsf{dyn}},\mathcal{Z}}(\kappa,\mathbf{x},z)\right)\\ &\quad\equiv \left(\text{MAL-VIEW}^{\mathsf{real},\mathsf{FaF}}_{\pi,\mathcal{A}_{\mathcal{L}^*}}(\kappa,\mathbf{x},(z_{\mathsf{mal}},(z_i)_{i:\rho_i\in[r]})),\ \mathrm{OUT}^{\mathsf{real},\mathsf{FaF}}_{\pi,\mathcal{A}_{\mathcal{L}^*}}(\kappa,\mathbf{x},(z_{\mathsf{mal}},(z_i)_{i:\rho_i\in[r]}))\right),\end{aligned}$$

where $(\mathcal{L}, z_{\mathsf{mal}}, (z_i)_{i\in[n]}) \leftarrow \mathcal{Z}(z)$ and $\mathcal{L}^*$ is the partial corruption list of $\mathcal{L}$. Thus,

$$\mathrm{REAL}^{\mathsf{dyn}}_{\pi,\mathcal{A}_{\mathsf{dyn}},\mathcal{Z}}(\kappa,\mathbf{x},z) \equiv \mathrm{REAL}^{\mathsf{FaF}}_{\pi,\mathcal{A}_{\mathcal{L}^*},\mathcal{A}_{\mathcal{H},\mathcal{L}}}(\kappa,\mathbf{x},(z_{\mathcal{A}},(z_i)_{i:\rho_i\in[r]}),(z_{\mathcal{A}},(z_i)_{i:\rho_i=\mathsf{PEC}})).$$

Similarly, in the ideal world,

$$\mathrm{IDEAL}^{\mathsf{dyn}}_{f,\mathcal{S}_{\mathsf{dyn}},\mathcal{Z}}(\kappa,\mathbf{x},z) \equiv \mathrm{IDEAL}^{\mathsf{FaF}}_{f,\mathcal{S}_{\mathcal{L}^*},\mathcal{S}_{\mathsf{sh},\mathcal{L}}}(\kappa,\mathbf{x},(z_{\mathsf{mal}},(z_i)_{i:\rho_i\in[r]}),(z_{\mathsf{mal}},(z_i)_{i:\rho_i=\mathsf{PEC}})).$$

By the assumed strong FaF security of π, the right-hand side of both equations are computationally indistinguishable. We conclude that the output of $\mathcal{Z}$ in both the real and ideal worlds is indistinguishable as well. □

Alon et al. [1] showed that if $t + h < n/2$, then any n-party functionality can be computed with statistical (t,h)-strong FaF security; and if $3t + 2h < 2$, then any n-party functionality can be computed with perfect (t,h)-strong FaF security. Conversely, they showed if the inequalities do not hold, then there exists an n-party functionality that cannot be computed with (t,h)-strong FaF security. Thus, as a corollary of Theorem 4.1, we obtain that the same holds for dynamic security.

Corollary 4.2. *Let $t, h, n \in \mathbb{N}$ be such that $t + h < n/2$ and let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality. Then f can be computed with statistical (t,h)-dynamic security. Conversely, if $t + h \geq n/2$, then there exists an n-party functionality that cannot be computed with statistical (t,h)-dynamic security.*

Corollary 4.3. *Let $t, h, n \in \mathbb{N}$ be such that $3t + 2h < n$ and let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality. Then f can be computed with perfect (t,h)-dynamic security. Conversely, if $3t + 2h \geq n$, then there exists an n-party functionality that cannot be computed with perfect (t,h)-dynamic security.*

In Section 5, we present and prove the corresponding statement for the computational setting.

5 Characterizing Computational Dynamic Security

In this section, we show that if $2t + h < n$ then any n-party functionality can be computed with computational (t,h)-dynamic security, and if $2t + h \geq n$ then there exists a n-party functionality that cannot be computed with (t,h)-dynamic security.

Theorem 5.1. *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality and let $t, h \in [n]$ be such that $2t + h < n$. Assume the existence of simulatable public-key encryption, collision-resistant hash functions, trapdoor permutations, augmented non-committing encryption, and dense cryptosystems. Then there exists an n-party protocol computing f with computational (t,h)-dynamic security. Conversely, if $2t + h \geq n$, then there exists an n-party functionality that cannot be computed with computational (t,h)-dynamic security.*

The proof is given below in Section 5.3. We first present the idea of the proof. By Theorem 4.1 it suffices to prove the result for the simpler strong FaF security notion. In particular, this solved an open problem left by [1] about the feasibility of computational strong FaF security. When $2t+h \geq n$, Alon et al. [1] showed the existence of an n-party functionality that cannot be computed with (t,h)-strong FaF security (in fact, they showed the impossibility for the weaker notion of (t,h)-FaF security).

We now consider the case where $2t + h < n$. We construct a secure protocol in 3 steps. First, we start with the protocol of Garg and Sahai [18]. Their protocol computes any functionality while guaranteeing security-with-abort against any adaptive adversary corrupting at most $n - 1$ parties in total. We note that their protocol actually satisfies a stronger requirement: in case of abort, all honest parties agree on a pair $\{\mathsf{P}_i, \mathsf{P}_j\}$ such that at least one of them is corrupted by the adversary. Since adaptive security implies strong FaF security, their protocol has this property for strong FaF security. We call this notion *security-with-semi-identifiable-abort*. See Section 5.1 for a formal definition. For the second step, we show how for inputless functionalities, security-with-semi-identifiable-abort can be augmented to full security (assuming $2t + h < n$). Finally, for the third step, we use the protocol of Melissaris, Ravi, and Yakoubov [22], which admits strong FaF (full) security assuming the parties are given correlated randomness.

In Section 5.1, we formally define the security that the Garg and Sahai protocol achieves. Then, in Section 5.2, we show how to generate correlated randomness with strong FaF security. Finally, in Section 5.3, we put together all the results to prove Theorem 5.1.

5.1 Strong FaF Security-With-Semi-Identifiable-Abort

We next define strong FaF security-with-semi-identifiable-abort, which roughly states that either all honest parties obtain the output, or they agree on a pair of parties such that at least one of them is malicious. Formally, we define security by describing an appropriate ideal world. Unlike the ideal world for full security, here the adversary is allowed to abort the computation *after* it learns the output; however, this comes at the cost of revealing the identity of two parties, one of which is malicious. In addition, since there is no guarantee that in the real-world the semi-honest parties won't learn the output, we always let the semi-honest parties to receive their output in the ideal execution.

We next formalize the ideal world. The computation is parameterized by a (potentially randomized) n-party functionality $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ to compute. An ideal computation of f on input $\mathbf{x} = (x_1, \ldots, x_n)$ with security parameter κ in the presence of a malicious adversary (a simulator) $\mathcal{S}_{\mathsf{mal}}$ corrupting $\mathcal{I}$, and a semi-honest adversary $\mathcal{S}_{\mathsf{sh}}$ corrupting $\mathcal{H}$, proceeds as follows.

Inputs: Each party P_i holds 1^κ and a private input $x_i \in \{0,1\}^*$. The adversaries $\mathcal{S}_{\mathsf{mal}}$ and $\mathcal{S}_{\mathsf{sh}}$ are given auxiliary inputs $z_{\mathsf{mal}} \in \{0,1\}^*$ and $z_{\mathsf{sh}} \in \{0,1\}^*$, respectively, and the private input of every party controlled by them. The trusted party T holds 1^κ.

Parties send inputs: Each uncorrupted party $\mathsf{P}_i \in \mathcal{P} \setminus \mathcal{I}$ sends x_i as its input to T. For each corrupted party, the malicious adversary $\mathcal{S}_{\mathsf{mal}}$ sends to T a value from its domain. In case the adversary does not send any input, the trusted party replaces its input with a default value. Write $(x'_1, \ldots, x'_n)$ for the tuple of inputs received by the trusted party.

The trusted party performs the computation: The trusted party T samples a random string rnd and computes $\mathbf{y} = (y_1, \ldots, y_n) = f(x'_1 \ldots, x'_n; \mathsf{rnd})$. It then sends $\mathbf{y}_{\mathcal{I}}$ to $\mathcal{S}_{\mathsf{mal}}$ and $\mathbf{y}_{\mathcal{H}}$ to $\mathcal{S}_{\mathsf{sh}}$.

The malicious adversary sends its (ideal-world) view: $\mathcal{S}_{\mathsf{mal}}$ sends to $\mathcal{S}_{\mathsf{sh}}$ its randomness, inputs, auxiliary input, and the outputs received from T.

The malicious adversary instructs the trusted party to continue or halt: The adversary $\mathcal{S}_{\mathsf{mal}}$ sends to T either $\mathsf{continue}$ or $(\mathsf{abort}, \mathsf{P}_i, \mathsf{P}_j)$ for some P_i and P_j such that at least one of them is in $\mathcal{I}$. If it sends $\mathsf{continue}$, then the trusted party sends y_k to P_k for every $k \in [n]$. Otherwise, if $\mathcal{S}_{\mathsf{mal}}$ sends $(\mathsf{abort}, \mathsf{P}_i, \mathsf{P}_j)$, then T sends $(\mathsf{abort}, \mathsf{P}_i, \mathsf{P}_j)$ to all parties.

Outputs: Each uncorrupted party (i.e., not in $\mathcal{I}$) outputs whatever it received from T (in particular, the parties in $\mathcal{H}$ output $(\mathsf{abort}, \mathsf{P}_i, \mathsf{P}_j)$ if they received it in the last step), the parties in $\mathcal{I}$ output nothing, and both $\mathcal{S}_{\mathsf{mal}}$ and $\mathcal{S}_{\mathsf{sh}}$ output some function of their respective view.

We next define the global view for the above ideal world. For inputs $\mathbf{x} = (x_1, \ldots, x_n)$, security parameter $\kappa \in \mathbb{N}$, and adversaries $\mathcal{S}_{\mathsf{mal}}$ and $\mathcal{S}_{\mathsf{sh}}$ controlling the parties in $\mathcal{I} \subseteq \mathcal{P}$ and $\mathcal{H} \subseteq \mathcal{P} \setminus \mathcal{I}$, and holding auxiliary inputs z_{mal} and z_{sh}, respectively, we define the following. Let $\mathcal{S}_{\mathsf{FaF}} = (\mathcal{S}_{\mathsf{mal}}, \mathcal{S}_{\mathsf{sh}})$, let $z_{\mathsf{FaF}} = (z_{\mathsf{mal}}, z_{\mathsf{sh}})$, and let $\text{IDEAL}^{\mathsf{swsia},\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{FaF}}}(\kappa, \mathbf{x}, z_{\mathsf{FaF}})$ denote the joint outputs of $\mathcal{S}_{\mathsf{mal}}$, $\mathcal{S}_{\mathsf{sh}}$, and the uncorrupted parties (i.e., those in $\mathcal{P} \setminus \mathcal{I}$), in a random execution of the above ideal-world process when running alongside $\mathcal{S}_{\mathsf{mal}}$.

Having defined the real and ideal models, we can now define strong FaF security-with-semi-identifiable-abort according to the real vs. ideal paradigm.

Definition 5.2 (Strong FaF security-with-semi-identifiable-abort)**.** *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality, and let π be a protocol computing f. We say that π computes f with* computational (t,h)-strong FaF security-with-semi-identifiable-abort*, if the following holds. For every malicious* PPT *adversary $\mathcal{A}_{\mathsf{mal}}$ controlling a set $\mathcal{I} \subseteq \mathcal{P}$ of size at most t in the real world, there exists a* PPT *adversary (called simulator) $\mathcal{S}_{\mathsf{mal}}$ controlling $\mathcal{I}$ in the ideal world; and for every subset of the remaining parties $\mathcal{H} \subseteq \mathcal{P} \setminus \mathcal{I}$ of size at most h controlled by a* semi-honest PPT *adversary $\mathcal{A}_{\mathsf{sh}}$, there exists a* PPT *adversary $\mathcal{S}_{\mathsf{sh}}$ controlling $\mathcal{H}$ in the ideal world, such that*

$$\left\{\mathrm{IDEAL}^{\mathsf{swsia},\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{FaF}}}(\kappa,\mathbf{x},z_{\mathsf{FaF}})\right\}_{\kappa\in\mathbb{N},\mathbf{x}\in(\{0,1\}^*)^n,z_{\mathsf{FaF}}\in(\{0,1\}^*)^2} \stackrel{\mathrm{C}}{\equiv} \left\{\mathrm{REAL}^{\mathsf{FaF}}_{\pi,\mathcal{A}_{\mathsf{FaF}}}(\kappa,\mathbf{x},z_{\mathsf{FaF}})\right\}_{\kappa\in\mathbb{N},\mathbf{x}\in(\{0,1\}^*)^n,z_{\mathsf{FaF}}\in(\{0,1\}^*)^2}.$$

The Security-With-Semi-Identifiable-Abort Hybrid Model

The *hybrid model* is a model that extends the real model with a trusted party that provides ideal computation for specific functionalities. We will only be interested in the security-with-semi-identifiable-abort hybrid model, where the parties communicate with this trusted party in exactly the same way as in the ideal model described above.

Let f be a functionality. Then, an execution of a protocol π computing a functionality g in the (swsia, f)-hybrid model involves the parties sending normal messages to each other (as in the real model) and in addition, having access to a trusted party computing f in the secure-with-semi-identifiable-abort FaF ideal model. It is essential that the invocations of f are done sequentially, meaning that before an invocation of f begins, the preceding invocation of f must finish. In particular, there is at most a single call to f per round, and no other messages are sent during any round in which f is called.

The sequential composition theorem of Canetti [6], Alon et al. [1] for strong FaF security states the following. Let π_f be a protocol that securely computes f in the secure-with-semi-identifiable-abort ideal model. Then, if a protocol π_g computes g in the (f,swsia)-hybrid model, then the protocol $\pi_g^{\pi_f}$, that is obtained from π_g by replacing all ideal calls to the trusted party computing f with the protocol π_f, securely computes g in the real model.

Theorem 5.3 ([6, 1])**.** *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality, let π_f be a protocol that computes f with computational (t,h)-strong FaF security-with-semi-identifiable-abort, and let π_g be a protocol that computes g with (t,h)-strong FaF security in the (f,swsia)-hybrid model. Then, protocol $\pi_g^{\pi_f}$ computes g with (t,h)-strong FaF security in the real model.*

5.2 Generating Correlated Randomness with Strong FaF Security

In this section, we show how to generate correlated randomness with (t,h)-strong FaF (full) security assuming $2t+h<n$.

Theorem 5.4. *Let $f : \{\lambda\}^n \to (\{0,1\}^*)^n$ be an inputless n-party functionality, and let $t,h \in \mathbb{N}$ be such that $2t+h<n$. Then, assuming collision-resistant hash functions, trapdoor permutations, augmented non-committing encryption, and dense cryptosystems, there exists an n-party protocol computing f with (t,h)-strong FaF security.*

The proof is given below. We first sketch the idea of the construction. We let the parties jointly compute a secret sharing of the output f, where each output is signed with respect to

a signature scheme's secret key. The parties additionally receive the public key of the signature scheme. This is done using the protocol of Garg and Sahai [18], which admits adaptive security-with-semi-identifiable-abort (and hence strong FaF security-with-semi-identifiable-abort [1]). If the computation follows through, the parties reconstruct the output (after verifying each share using the signature's scheme's public key). Otherwise, they have the identity of two parties, where one of them is malicious. The parties remove them from the computation (even if one of them is honest) and restart the computation, this time with the shares of the removed parties shared among the remaining parties. If the computation follows through, then the parties send to the removed parties the signature scheme's public key and the shares of the removed parties' shares (as well as reconstructing the output for themselves). We show that since $2t+h < n$, the removed honest party (if such exists) can reconstruct its share. If the computation fails again, the parties will restart with two more parties removed. Since there is an honest majority, this process will either end with only honest parties remaining or with one of the intermediate computations successfully generating the correct correlated randomness. We next formalize this intuition. We use the following results due to Garg and Sahai [18] and Alon et al. [1].

Theorem 5.5 ([18, Theorem 3], implicit)**.** *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality and let $t, h \in [n]$ be such that $t+h < n$. Then, assuming collision-resistant hash functions, trapdoor permutations, augmented non-committing encryption, and dense cryptosystems, there exists an n-party protocol computing f with computational (t,h)-adaptive security-with-semi-identifiable-abort.*

Theorem 5.6 ([1, Theorem 5.3], implicit)**.** *For any $n, t, h \in \mathbb{N}$ and any n-party functionality $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$, if a protocol π computes f with (t,h)-adaptive security-with-semi-identifiable-abort, then it computes f with (t,h)-strong FaF security-with-semi-identifiable-abort.*

We state a lemma that asserts that f can be computed with strong FaF security in a hybrid model that provides the parties with the computation of the signed shares. The proof of Theorem 5.4 then follows from Theorems 5.5 and 5.6 and the composition theorem (Theorem 5.3). Before stating the lemma, let us define the functionalities that provide these. Let $\mathsf{Sig} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme and denote $\mathcal{N} = \{n-2t, n-2t+2, \ldots, n\}$. For a set $\mathcal{T} \subseteq [n]$ of size $|\mathcal{T}| \in \mathcal{N}$ define the inputless $|\mathcal{T}|$-party functionality $f_{\mathcal{T}}$ as follows.

Functionality 5.7 ($f_{\mathcal{T}}$)**.**

1. *Sample $\mathbf{y} \leftarrow f(\lambda, \ldots, \lambda)$ and $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa)$.*
2. *For every $i \in [n]$, compute shares of y_i in an t'-out-of-$|\mathcal{T}|$ Shamir's secret-sharing scheme among the parties in $\mathcal{T}$, where $t' = t + h + 1 - (n - |\mathcal{T}|)/2$ upper bounds the total number of remaining parties controlled by one of the two adversaries (observe that $t' \leq |\mathcal{T}|$ if and only if $2t + h < n$). For every $j \in \mathcal{T}$, let $y_i[j]$ denote the share of y_i of party P_j.*
3. *For every $i \in [n]$, sign each share of y_i: for every $j \in \mathcal{T}$, let $\sigma_{i,j} \leftarrow \mathsf{Sign}_{\mathsf{sk}}(y_i[j])$.*
4. *For every $j \in \mathcal{T}$, party P_j receives the public key pk, the shares $(y_i[j])_{i\in[n]}$, and their signatures $(\sigma_{i,j})_{i\in[n]}$.*

To prove Theorem 5.4, it suffices to prove the following (note that the assumptions in Theorem 5.4 imply the existence of signature schemes assumed in the lemma).

Lemma 5.8. *Let $f : \{\lambda\}^n \to (\{0,1\}^*)^n$ be an inputless n-party functionality and let $t, h \in [n]$ be such that $2t + h < n$. Then, assuming the existence of signature schemes, there exists an n-party protocol computing f with (t,h)-strong FaF security in the $\{(f_{\mathcal{T}}, \mathsf{swsia})\}_{\mathcal{T}\subseteq[n],|\mathcal{T}|\in\mathcal{N}}$-hybrid model.*

Proof. The protocol for computing f proceeds as follows.

Protocol 5.9 (π)**.**

1. *Let $\mathcal{T} = [n]$.*
2. *The parties call $(f_{\mathcal{T}}, \mathsf{swsia})$.*
3. *If the parties received $(\mathsf{abort}, \mathsf{P}_i, \mathsf{P}_j)$, then the parties update $\mathcal{T}$ to $\mathcal{T} \setminus \{\mathsf{P}_i, \mathsf{P}_j\}$ and go back to Step 2. Note that this can occur at most $t+1$ times.*
4. *Otherwise, every party P_j, where $j \in \mathcal{T}$, receives the public key pk, the shares $(y_i[j])_{i\in[n]}$, and their signatures $(\sigma_{i,j})_{i\in[n]}$.*
5. *Each party P_j, where $j \in \mathcal{T}$, broadcasts pk, and for every $i \in [n]$, sends $(y_i[j], \sigma_{i,j})$ to P_i.*
6. *Every party P_i sets pk' to be the majority of public keys it received. It then uses all shares $y_i[j]$ such that $\mathsf{Verify}_{\mathsf{pk}'}(y_i[j], \sigma_{i,j}) = 1$ to reconstruct y_i and output it.*

We now show the (t,h)-strong FaF security of the protocol. Let $\mathcal{A}_{\mathsf{mal}}$ be a malicious adversary controlling a set of parties $\mathcal{I} \subseteq \mathcal{P}$ of size $|\mathcal{I}| \leq t$. We define its simulator $\mathcal{S}_{\mathsf{mal}}$ as follows.

1. Receive $\mathbf{y}_{\mathcal{I}}$ from the trusted party (recall that the functionality has no inputs).
2. Let $\mathcal{T} = [n]$.
3. Emulate the call to $(f_{\mathcal{T}}, \mathsf{swsia})$:
   (a) Sample $(\mathsf{pk}, (y_i[j], \sigma_{i,j})_{i\in[n],j\in\mathcal{T}}) \leftarrow f_{\mathcal{T}}(\lambda, \ldots, \lambda)$.
   (b) Send $(\mathsf{pk}, (y_i[j], \sigma_{i,j})_{i\in[n],j\in(\mathcal{T}\cap\mathcal{I})})$ to $\mathcal{A}_{\mathsf{mal}}$.
   (c) If $\mathcal{A}_{\mathsf{mal}}$ responds with $(\mathsf{abort}, \mathsf{P}_i, \mathsf{P}_j)$, then update $\mathcal{T}$ to $\mathcal{T} \setminus \{\mathsf{P}_i, \mathsf{P}_j\}$ and go back to Step 3a.
4. For every $i \in \mathcal{I}$, complete the shares $(y_i[j])_{j\in(\mathcal{T}\cap\mathcal{I})}$ from the last emulation of $(f_{\mathcal{T}}, \mathsf{swsia})$ to shares of y_i (this is possible due to the properties of the Shamir's secret sharing scheme), and sign them using pk.
5. Send pk and the signed shares of each y_i for all $i \in \mathcal{I}$, to $\mathcal{A}_{\mathsf{mal}}$, output whatever $\mathcal{A}_{\mathsf{mal}}$ outputs, and halt.

Next, fix a semi-honest adversary $\mathcal{A}_{\mathsf{sh}}$ controlling a set of parties $\mathcal{H} \subseteq \mathcal{P} \setminus \mathcal{I}$ of size $|\mathcal{H}| \leq h$. We define its simulator $\mathcal{S}_{\mathsf{mal}}$ as follows.

1. Receive $\mathbf{y}_{\mathcal{H}}$ from T and receive the randomness and auxiliary input of $\mathcal{S}_{\mathsf{mal}}$.
2. Run $\mathcal{S}_{\mathsf{mal}}$ using its randomness and auxiliary input to compute all samples $(\mathsf{pk}, (y_i[j], \sigma_{i,j})_{i\in[n],j\in\mathcal{T}})$ of $f_{\mathcal{T}}$.
3. Query $\mathcal{A}_{\mathsf{mal}}$ on each $(\mathsf{pk}, (y_i[j], \sigma_{i,j})_{i\in[n],j\in(\mathcal{T}\cap\mathcal{I})})$ to receive the non-prescribed messages that $\mathcal{A}_{\mathsf{mal}}$ sent to the parties in $\mathcal{H}$ after each call to $(f_{\mathcal{T}}, \mathsf{swsia})$.
4. For every $i \in \mathcal{H}$, complete the shares $(y_i[j])_{j\in(\mathcal{T}\cap\mathcal{H})}$ from the last emulation of $(f_{\mathcal{T}}, \mathsf{swsia})$ to shares of y_i and sign them.
5. Send to $\mathcal{A}_{\mathsf{sh}}$ all the samples $(\mathsf{pk}, (y_i[j], \sigma_{i,j})_{i\in[n],j\in(\mathcal{T}\cap\mathcal{H})})$ of $f_{\mathcal{T}}$, the completed shares with their signatures, and the non-prescribed messages of $\mathcal{A}_{\mathsf{mal}}$, and all signed shares of each y_i for all $i \in \mathcal{H}$. Output whatever $\mathcal{A}_{\mathsf{sh}}$ outputs and halt.

We next prove that

$$\left\{\mathrm{IDEAL}^{\mathsf{FaF}}_{f,\mathcal{S}_{\mathsf{FaF}}}(\kappa, \mathbf{x}, z_{\mathsf{FaF}})\right\}_{\kappa\in\mathbb{N},\mathbf{x}\in(\{0,1\}^*)^n,z_{\mathsf{FaF}}\in(\{0,1\}^*)^2} \tag{1}$$
$$\stackrel{\mathrm{C}}{\equiv} \left\{\mathrm{REAL}^{\mathsf{FaF}}_{\pi,\mathcal{A}_{\mathsf{FaF}}}(\kappa, \mathbf{x}, z_{\mathsf{FaF}})\right\}_{\kappa\in\mathbb{N},\mathbf{x}\in(\{0,1\}^*)^n,z_{\mathsf{FaF}}\in(\{0,1\}^*)^2}.$$

First, note that in every iteration of Step 2, the total number of shares held by $\mathcal{A}_{\mathsf{mal}}$ and $\mathcal{A}_{\mathsf{sh}}$ is smaller than the secret-sharing threshold. Indeed, for every $k \in \{1, \ldots, t+1\}$, in the k^{th} iteration there are at most $t-k+1$ corrupted parties (i.e., in $\mathcal{I}$), at most h semi-honest parties, and a total of $|\mathcal{T}| = n - 2(k-1)$ parties. Since the threshold is

$$t + h + 1 - \frac{n - |\mathcal{T}|}{2} = t + h + 2 - k > (t - k + 1) + h,$$

by the privacy property of the secret-sharing scheme, at every iteration, the joint view of the two adversaries consists of only random independent shares. In particular, except for the last iteration where the output is reconstructed, all signed shares are independent of the output. Now, since the same holds in the ideal world, it follows that $\mathcal{A}_{\mathsf{mal}}$ aborts in the real world if and only if it aborts in the ideal world. Moreover, it responds with the same pair of parties to abort. As for the last iteration, the properties of Shamir's secret-sharing scheme allow the simulators to complete the shares of $\mathcal{A}_{\mathsf{mal}}$ and $\mathcal{A}_{\mathsf{sh}}$ to shares of the output given by the trusted party, implying Equation (1) as required.

□

5.3 Putting it all Together

We now combine Theorem 5.4 with known results to prove Theorem 5.1. We will use the following results due to Alon et al. [1] and Melissaris et al. [22].

Theorem 5.10 ([1, Theorem 4.2])**.** *For all $n, t, h \in \mathbb{N}$ such that $2t + h \geq n$, there exists an n-party functionality that cannot be computed with computational (t, h)-strong FaF security.*

Theorem 5.11 ([22, Theorem 9])**.** *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality and let $t, h \in [n]$ be such that $2t + h < n$. Then, assuming the existence of simulatable public-key encryption and dense cryptosystems, there exists an n-party protocol in the correlated randomness model computing f with computational (t, h)-strong FaF security.*

We next prove our main result of the section.

Proof of Theorem 5.1. For the first direction, assume that $2t + h < n$ and fix an n-party functionality f. By Theorem 5.4, any inputless n-party functionality can be computed with computational (t, h)-strong FaF security. Therefore, by Theorems 5.3 and 5.11, there exists an n-party protocol computing f with computational (t, h)-strong FaF security. Finally, by Theorem 4.1 this protocol is also computationally (t, h)-dynamic secure.

For the second direction, assume that $2t + h \geq n$. By Theorem 5.10, there exists an n-party functionality that cannot be computed with computational (t, h)-strong FaF security. Thus, by Theorem 4.1 it cannot be computed with computational (t, h)-dynamic security. □

Acknowledgements

We are grateful to Eran Omri, Amos Beimel, and Muthuramakrishnan Venkitasubramaniam for many helpful discussions.

Bibliography

[1] B. Alon, E. Omri, and A. Paskin-Cherniavsky. MPC with friends and foes. In D. Micciancio and T. Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 677–706. Springer, 2020.

[2] B. Alon, A. Beimel, and E. Omri. Three party secure computation with friends and foes. In G. N. Rothblum and H. Wee, editors, *Theory of Cryptography - 21st International Conference, TCC 2023, Taipei, Taiwan, November 29 - December 2, 2023, Proceedings, Part II*, volume 14370 of *Lecture Notes in Computer Science*, pages 156–185. Springer, 2023.

[3] G. Asharov, R. Cohen, and O. Shochat. Static vs. adaptive security in perfect MPC: A separation and the adaptive security of BGW. In D. Dachman-Soled, editor, *3rd Conference on Information-Theoretic Cryptography, ITC 2022, July 5-7, 2022, Cambridge, MA, USA*, volume 230 of *LIPIcs*, pages 15:1–15:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[4] D. Beaver and S. Haber. Cryptographic protocols provably secure against dynamic adversaries. In R. A. Rueppel, editor, *Advances in Cryptology - EUROCRYPT '92, Workshop on the Theory and Application of of Cryptographic Techniques, Balatonfüred, Hungary, May 24-28, 1992, Proceedings*, volume 658 of *Lecture Notes in Computer Science*, pages 307–323. Springer, 1992.

[5] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for noncryptographic fault-tolerant distributed computations. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–10, 1988.

[6] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[7] R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively secure multi-party computation. In G. L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 639–648. ACM, 1996.

[8] R. Canetti, I. Damgård, S. Dziembowski, Y. Ishai, and T. Malkin. On adaptive vs. non-adaptive security of multiparty protocols. In B. Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, volume 2045 of *Lecture Notes in Computer Science*, pages 262–279. Springer, 2001.

[9] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In J. H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 494–503. ACM, 2002.

[10] R. Canetti, O. Poburinnaya, and M. Venkitasubramaniam. Equivocating yao: constant-round adaptively secure multiparty computation in the plain model. In H. Hatami, P. McKenzie, and V. King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 497–509. ACM, 2017.

[11] D. Chaum, C. Crépeau, and I. Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 11–19, 1988. doi: 10.1145/62212.62214.

[12] A. R. Choudhuri, A. Goel, M. Green, A. Jain, and G. Kaptchuk. Fluid MPC: secure multiparty computation with dynamic participants. In T. Malkin and C. Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 94–123. Springer, 2021.

[13] R. Cohen, J. A. Garay, and V. Zikas. Completeness theorems for adaptively secure broadcast. In H. Handschuh and A. Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part I*, volume 14081 of *Lecture Notes in Computer Science*, pages 3–38. Springer, 2023.

[14] G. Deligios, A. Goel, and C. Liu-Zhang. Maximally-fluid MPC with guaranteed output delivery. *IACR Cryptol. ePrint Arch.*, page 415, 2023. URL https://eprint.iacr.org/2023/415.

[15] U. Feige. Noncryptographic selection protocols. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 142–152. IEEE, 1999.

[16] J. A. Garay, J. Katz, R. Kumaresan, and H. Zhou. Adaptively secure broadcast, revisited. In C. Gavoille and P. Fraigniaud, editors, *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, USA, June 6-8, 2011*, pages 179–186. ACM, 2011.

[17] J. A. Garay, Y. Ishai, R. Ostrovsky, and V. Zikas. The price of low communication in secure multi-party computation. In J. Katz and H. Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 420–446. Springer, 2017.

[18] S. Garg and A. Sahai. Adaptively secure multi-party computation with dishonest majority. In R. Safavi-Naini and R. Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 105–123. Springer, 2012.

[19] 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 the 51st Annual ACM STOC*, pages 218–229, 1987.

[20] M. Hirt and V. Zikas. Adaptively secure broadcast. In H. Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 466–485. Springer, 2010.

[21] Y. Ishai, A. Kumarasubramanian, C. Orlandi, and A. Sahai. On invertible sampling and adaptive security. In M. Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 466–482. Springer, 2010.

[22] N. Melissaris, D. Ravi, and S. Yakoubov. Threshold-optimal MPC with friends and foes. In A. Chattopadhyay, S. Bhasin, S. Picek, and C. Rebeiro, editors, *Progress in Cryptology - INDOCRYPT 2023 - 24th International Conference on Cryptology in India, Goa, India, December 10-13, 2023, Proceedings, Part II*, volume 14460 of *Lecture Notes in Computer Science*, pages 3–24. Springer, 2023.

[23] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 73–85, 1989.

[24] A. C. Yao. Protocols for secure computations (extended abstract). In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 160–164, 1982.