An Unstoppable Ideal Functionality for Signatures and a Modular Analysis of the Dolev-Strong Broadcast

Ran Cohen
cohenran@runi.ac.il
Reichman University

Jack Doerner
j@ckdoerner.net
Brown University

Eysa Lee
eysa_lee@brown.edu
Brown University

Anna Lysyanskaya
anna_lysyanskaya@brown.edu
Brown University

Lawrence Roy
ldr709@gmail.com
Aarhus University

December 17, 2024

Abstract

Many foundational results in the literature of consensus follow the Dolev-Yao model (FOCS '81), which treats digital signatures as ideal objects with *perfect* correctness and unforgeability. However, no work has yet formalized an ideal signature scheme that is both suitable for this methodology and possible to instantiate, or a composition theorem that ensures security when instantiating it cryptographically.

The Universal Composition (UC) framework would ensure composition if we could specify an ideal functionality for signatures and prove it UC-realizable. Unfortunately, *all* signature functionalities heretofore proposed are problematic when used to construct higher-level protocols: either the functionality internally computes a computationally secure signature, and therefore higher-level protocols must rely upon computational assumptions, or else the functionality introduces a *new* attack surface that does not exist when the functionality is realized. As a consequence, no consensus protocol has ever been analyzed in a modular way using existing ideal signature functionalities.

We propose a new *unstoppable* ideal functionality for signatures that is UC-realized exactly by the set of standard EUF-CMA signature schemes that are *consistent* and *linear time*. No adversary can prevent honest parties from obtaining perfectly ideal signature services from our functionality. We showcase its usefulness by presenting the first modular analysis of the Dolev-Strong broadcast protocol (SICOMP '83) in the UC framework. Our result can be interpreted as a step toward a sound realization of the Dolev-Yao methodology. We also generalize our result to the threshold setting.

Contents

1 Introduction **1**
1.1 Shortcoming of Directly Using EUF-CMA in Cryptographic Protocols . . . 2
1.2 The Evolution of Ideal Signature Formulations . . . 4
1.3 Our Contributions and Technical Overview . . . 7

2 Preliminaries **13**
2.1 Digital Signature Schemes . . . 13
2.2 The Universal Composability Framework . . . 15

3 An Unstoppable Signature Functionality **17**
3.1 The Functionality . . . 17
3.2 Equivalence to Consistent Linear-Time EUF-CMA . . . 19
3.3 Extracting a Signature Scheme from Any UC-Secure Signature Protocol . . . 27

4 A Modular Analysis of the Dolev-Strong Broadcast **33**
4.1 Modeling Synchronous Protocols in UC . . . 33
4.2 The Broadcast Functionality . . . 34
4.3 The Dolev-Strong Broadcast Protocol . . . 36
4.4 A Modular Proof of Security for Dolev-Strong . . . 38
4.5 Attacks on Dolev-Strong Broadcast under Prior Signature Functionalities . . . 42

5 Generalizing to the Threshold Setting **43**
5.1 A Fourth-Generation Threshold Signature Functionality . . . 44
5.2 Realizing $\mathcal{F}_{\mathsf{tsig}}$ via a Third-Generation Signing Functionality . . . 46
5.3 Proof of Security . . . 48
5.4 On Realizing Our Third-Generation Threshold Signature Functionality . . . 50

References **50**

A Prior Signature Functionalities **55**
A.1 The First Generation Signature Functionality of Canetti [Can04] . . . 55
A.2 The Second Generation Signature Functionality of Canetti [Can05] . . . 56

B Synchronous Protocols in UC (Continued) **56**

1 Introduction

Digital signatures are one of the most fundamental tools of modern cryptography. Using a digital signature scheme, Alice can send a message to Bob in a way that convinces him that Alice is indeed the sender. Bob can then transfer the signed message to Charlie, who will also be convinced that Alice is the source. Signatures were first introduced by Diffie and Hellman [DH76], constructed by Rivest, Shamir, and Adleman [RSA78], and formalized by Goldwasser, Micali, and Rivest [GMR88]. Today's standard security definition requires a signature scheme to satisfy two properties. The first, *correctness*, asserts that the scheme's verification algorithm considers messages that are honestly signed using honestly generated keys as valid.[1] The second, *existential unforgeability under chosen-message attacks* (EUF-CMA), asserts that a polynomial-time adversary that can obtain signatures on arbitrary messages of its choice (via a signing oracle) is able to forge a valid signature on a new message with at most negligible probability.

Digital signatures play a central role in many distributed consensus tasks such as *Byzantine agreement*, wherein n parties must agree on a single common value even when t of them collude and actively cheat, and such as *broadcast*, wherein the agreement is on the value of a designated, potentially cheating sender. The papers that originally introduced these problems [PSL80, LSP82] showed that no solution exists in the plain model (i.e., without setup assumptions) for $t \geq n/3$. However, given *digital signatures* and a *public-key infrastructure* (PKI, a setup phase in which parties reliably distribute their verification keys) a solution exists with optimal resilience against $t < n$ corrupted participants for broadcast, and $t < n/2$ for Byzantine agreement.

The solutions proposed by these early works [PSL80, LSP82] are exponential in the number of parties n, and thus only useful if n is logarithmic in the security parameter. The first *efficient* broadcast was introduced not long afterward by Dolev and Strong [DS83]; like its predecessors it assumes only the existence of digital signatures and a public key infrastructure. Further breakthrough results followed. For example, a line of works has proven that broadcast can be achieved with expected-constant round complexity in the honest-majority [KK06] and even *dis*honest-majority [GKKO07, FN09, WXSD20] settings. It remains true that when security against any arbitrary set of colluding parties is required, the only known broadcast protocols are based on Dolev-Strong [DS83]. These results supply important building blocks and inflential techniques for secure multiparty computation (MPC). A common design pattern in MPC research is to devise a protocol that uses an *ideal* broadcast channel, which is later replaced by an efficient broadcast protocol.

Existing proofs of Dolev-Strong [DS83] (and all other signature-based Byzantine agreement and broadcast protocols) treat digital signature schemes as *perfectly correct* and *perfectly unforgeable*, and do not present an explicit reduction from an adversary attacking the protocol to the security of the signature scheme. This stands in sharp contrast with the standard practice in cryptography, and with the modular spirit with which proofs are often structured in the literature of MPC. While it is typical to construct high-level protocols from ideal building blocks, it is also usually expected that those ideal building blocks can later be replaced by sub-protocols that *securely realize* them. The analysis of each sub-protocol is conducted by considering a single instance in a simplified setting, and a *composition theorem* is used to reach a conclusion about the security of high-level protocols when these sub-protocols are plugged in.[2] In essence, a composition theorem translates any adversary attacking a *composed* protocol (that uses real sub-protocols) into either an adversary attacking the corresponding *un*composed protocol (that uses ideal building blocks) or an adversary

[1]This might hold *always* in the case of perfect correctness, or it might hold with overwhelming probability.

[2]The nature of the required composition theorem depends upon the ways in which the sub-protocols are invoked; in general they may be invoked concurrently and by arbitrary sets of participants, but simpler composition theorems exist for restricted settings.

attacking a single instance of the real sub-protocol in isolation, with only a negligible difference in adversarial advantage.

For example, the classic MPC protocol of Goldreich, Micali, and Wigderson (GMW) [GMW87] is simple to formulate given an ideal broadcast channel and trusted parties that ideally compute oblivious transfer (OT) and zero-knowledge proofs. These ideal entities are known more generally as *ideal functionalities*. If the broadcast channel is realized by the Dolev-Strong protocol [DS83], then proving security requires translating every attack on the (composed) GMW protocol into an attack on the underlying signature scheme. Since attacks on secure digital signatures succeed with at most negligible probability by definition, such a proof ensures that an attack on the composed protocol can also succeed with at most negligible probability, or else the security of the signature scheme is invalidated. This pattern of describing MPC protocols using an ideal broadcast functionality, to be realized modularly, is found throughout the modern MPC literature, yet, as we have said, no formal means of invoking a reduction to the security of signatures when such a realization is performed has been supplied.

The most direct solution in the case of any *specific* broadcast protocol would be to write an explicit proof that the protocol realizes the broadcast functionality, with a reduction to the correctness and EUF-CMA properties of the underlying signature scheme. This solution has a number of downsides, which we discuss in Section 1.1. The alternative is to construct a notion of ideal signatures that is *realizable* by standard EUF-CMA signatures while also being *compatible* with existing broadcast protocol proofs and with whichever composition theorem the higher-level protocol requires.

Devising an ideal functionality that captures the security of digital signatures is a challenging task, and numerous attempts have been made, starting with the work of Canetti [Can01]. Such ideal functionalities must be realizable by all *reasonable* signatures[3] on the one hand, and support a modular analysis of cryptographic protocols on the other. As we argue in Section 1.2, *all* previous formulations of an ideal functionality for digital signatures either rely on computational assumptions in the ideal computation (which results in a cumbersome proof), or introduce a means for the adversary to *block* honest parties from issuing signatures when the functionality is used in a hybrid model; this denial-of-service attack translates into an attack on the security of higher-level protocols such as the Dolev-Strong protocol, as we show explicitly in Section 4.5.

The main contribution of this work is an *unstoppable* ideal functionality for signatures, which on the one hand is realized by all EUF-CMA signatures (that satisfy a few additional properties) and on the other hand can never be blocked, thus preventing denial-of-service attacks by an adversary when it is used by another protocol. We showcase the usefulness of our unstoppable functionality by providing the first modular analysis of Dolev-Strong broadcast [DS83]. We also generalize our result to the threshold setting.

1.1 Shortcoming of Directly Using EUF-CMA in Cryptographic Protocols

We begin by emphasizing that while our work gives a modular proof of Dolev-Strong [DS83] (and can be used in a similar way to prove similar protocols), we do not claim that there are any inherent flaws in the protocol as it exists in the literature currently. The gap we identify lies in the *proofs* of such protocols and not in the constructions themselves.

Before we describe prior ideal signature functionality formulations (and their shortcomings), we note that using the correctness and unforgeability properties of signature schemes directly in the

[3] We leave the qualifier somewhat ambiguous for now; our own functionality is realized by the set of EUF-CMA signatures that satisfy a few natural additional properties, which we introduce in Section 1.3 and formalize in Section 2.

analysis of high-level protocols carries several disadvantages:

- The resulting security proofs must be *monolithic* and are likely to be more complex than modular ones: if protocol analysis is property-based, i.e. it directly uses specific properties of a signature scheme and shows that they are violated if the higher-level protocol fails, then the underlying protocol cannot be decomposed into a "signature module" and a "high-level module" that uses the signature.

- Many lower bounds on broadcast and Byzantine agreement protocols apply only to *deterministic* protocols: for example, bounds on the round complexity [DS83] and communication complexity [DR82]. A protocol that explicitly uses signatures *cannot* be deterministic, and must admit a negligible (yet positive) error probability. As a result, the aforementioned lower bounds do not apply to such a protocol,[4] and we cannot satisfy ourselves that such a protocol is optimal unless new lower bounds are developed or the existing ones are enhanced to capture explicit signature use.

- The property-based EUF-CMA signature definition [GMR88, Gol04, KL14], now considered to be standard, does not explicitly address several security considerations that are important in the MPC context, such as *adversarially malformed keys*, *adaptive corruptions*, and *concurrent instances*.

- When property-based definition is used within a cryptographic protocol, it can be unclear which properties are required of the definition in order to achieve desired security guarantees for the protocol. This stands is contrast with a simulation-based definition, which is holistic in nature and captures all security requirements at once.

We exemplify the subtlety of determining the exact set of properties that we require of a signature scheme by considering adversarially malformed keys. EUF-CMA signatures are generally allowed to have probabilistic verification algorithms (e.g., the signature scheme of Boneh and Franklin [BF01]). When the verification algorithm is used with an honestly generated public key, the correctness property ensures that an honestly generated signature (using the corresponding secret key) will be accepted, and the EUF-CMA property ensures that signatures generated without access to the secret key will be rejected. However, there are *no guarantees* for maliciously generated public keys and signatures: the verification algorithm might provide inconsistent results for the same message/signature pair, which can be disastrous. In Section 4.5, we describe an explicit attack against the Dolev-Strong protocol [DS83] in the absence of guarantees about malicious keys and signatures.

This shortcoming was previously addressed by Canetti [Can04], who defined a notion of "weak consistency" which ensures that when the public key is honestly generated, the verification of the same message/signature pair will return the same value with all but negligible probability (as pointed out by Canetti, this form of consistency is implied by the non-repudiation requirement in

[4]We illustrate this with a contrived example. Given an EUF-CMA secure signature scheme, consider a new scheme that augments the signing key with a long random bit string during key generation, which is never used elsewhere. Clearly, this scheme is also EUF-CMA secure. Now consider the randomized protocol of Katz and Koo [KK06], and "de-randomize" it by instructing the parties to use the otherwise-unused random strings within their signing keys instead of tossing coins. The resulting protocol is "deterministic" except for the key-generation phase, yet it terminates after polylogarithmically many rounds (in fact, after and expected-constant number of rounds) and has a negligible error probability, assuming an honest majority. This illustrates that the lower bound of $t + 1$ rounds [DS83] does not apply to protocols that are deterministic except for the use of signatures.

[GMR88]). This notion was strengthened by Garay, Kiayias, and Zhou [GKZ10] to capture maliciously generated keys. Our signature functionality also guarantees this stronger form consistency: when verification is performed on the same input repeatedly, it always returns the same output.

The *correctness* property is also subtle, when it allows for a negligible error probability. For example, the definition in [GKZ10] ensures correctness with all but negligible probability for messages chosen before the signing key is sampled. Under this definition, it may not be possible for a signer to sign their own public key. On the other hand, our definition requires correctness for messages generated adaptively by an adversary with knowledge of the public key and all previous signatures.

As of now, the only proof that we are aware of that explicitly reduces the security of *any* broadcast protocol to the EUF-CMA security of signatures is by Lindell, Lysyanskaya, and Rabin [LLR02], who proved security under sequential composition for a very simple broadcast protocol among three participants. Their proof establishes only a property-based notion of broadcast, rather than the realization of an ideal broadcast functionality under arbitrary composition that we seek. Nevertheless, it illustrates the tediousness of formalizing such arguments even when the protocol is much simpler than the Dolev-Strong protocol and the proof intuition is straightforward. In contrast, the modular approach that we take in Section 4 yields a simpler proof with a strong resemblance to the classic proof of Dolev-Strong.

1.2 The Evolution of Ideal Signature Formulations

The Dolev-Yao Model. In the Dolev-Yao model [DY81], cryptographic primitives such as encryption and digital signatures are considered as abstract symbolic operations with perfect security. This methodology is common in the literature of broadcast and Byzantine agreement; it was used in the original broadcast papers of the 1980s [PSL80, LSP82, DS83], and it has regained popularity in recent years [BKL19, WXSD20, FLL21, GGL22, LL22, GLW22, TLP22, LN24, ELP24]. In all of these works, signatures are considered to be perfectly correct and perfectly unforgeable; this approach has recently been pushed even further with works assuming perfectly secure *threshold* signatures [FLL21] and perfectly secure *unique* threshold signatures [GGL22].

Use of the Dolev-Yao model in a security proof carries the hope (at least implicitly) that the proven protocol remains secure when the ideal signatures within it are replaced by EUF-CMA signatures. However, this hope faces two formal shortcomings: First, the model does not include a formal definition for ideal signatures (or ideal public keys), and second, if there were such a definition, then there would still be no composition theorem to ensure security is retained when the ideal signatures are instantiated cryptographically. These shortcomings lead to several drawbacks when the Dolev-Yao model is used in security proofs.

First, Dolev-Yao is not a sound methodology in the sense that it is possible to construct protocols that are secure in the Dolev-Yao model, but insecure when EUF-CMA signatures are used instead. For example, the protocols of Pease, Shostak, and Lamport [PSL80, LSP82] are perfectly secure in the Dolev-Yao model, but their complexity is exponential in the number of parties, and thus they are not secure when EUF-CMA signatures are used and the number of parties is super-logarithmic. Similarly, it is easy to construct inductive arguments in the Dolev-Yao model (see, for example, Fitzi et al. [FLL21]), but the soundness of such arguments can fail if even a negligible security loss is introduced in each inductive step [Lin19].

Second, it is not always clear how one can translate an attack on a protocol in the Dolev-Yao model into an attack on the concrete signature scheme used to instantiate the model. This stands in sharp contrast to the standard security paradigm for MPC protocols, which would demand the existence of an explicit reduction.

Third, the assumption that an adversary *cannot* forge a signature severely limits the adversarial strategies that are formally captured by the security proof. When signatures are represented as bit-strings, an adversary has a positive probability of forging a signature if it deviates from the protocol specification in *any* way, even if it does not *intend* to forge. It is not even necessarily the case that the adversary can determine whether it has forged or not (e.g., if the verification key is not public). It follows that an adversary can only be sure that it will never forge a signature if it follows the protocol specification exactly (and potentially causes some parties to crash). The Dolev-Yao model guarantees little if any security against adversaries that deviate from protocol instructions in an arbitrary way.

Supplying a Composition Theorem. Canetti's Universal Composability (UC) framework [Can01] provides the composition theorem that the Dolev-Yao model lacks. Informally, it guarantees that if a cryptographic scheme *UC-realizes* an ideal signature functionality $\mathcal{F}_{\mathsf{sig}}$, then that scheme can be used to replace $\mathcal{F}_{\mathsf{sig}}$ in any protocol that uses $\mathcal{F}_{\mathsf{sig}}$ as a subroutine. So, if a protocol π is secure (or has some other useful property) in an idealized model that is enhanced with $\mathcal{F}_{\mathsf{sig}}$, then it remains secure (and retains its other properties) when $\mathcal{F}_{\mathsf{sig}}$ is replaced by the cryptographic scheme. This leaves open a critical question:

> What formulation of $\mathcal{F}_{\mathsf{sig}}$ is both *useful* for designing the outer protocol π, and also UC-realizable by all EUF-CMA signatures?

The First Generation of $\mathcal{F}_{\mathsf{sig}}$. We identify three prior approaches to the design of signature functionalities, which are differentiated by the method that is used to generate the string representations of signatures and public keys. The oldest approach first appeared in the original version of the UC paper [Can01], and has also appeared in numerous follow-up works [Can04, CR03, BH04, GKZ10, CSV16, BCH+20]. In this approach, every key-generation or signing request causes the functionality to ask the *adversary* to provide a public key or signature string, respectively. The benefit of this formulation is that the ideal-model adversary is granted a great deal of power, and so the task of simulating real-world adversaries is relatively simple. However, the power given to the adversary is also the shortcoming of this formulation: it enables attacks on higher-level protocols that use this version of $\mathcal{F}_{\mathsf{sig}}$ as a subroutine, as we will now illustrate.

When a signing request is received, the functionality passes the activation token to the adversary in order allow it to compute a signature string. This implies that the adversary learns when each signing attempt occurs, and learns on which message a signature has been requested, even if the signer is *honest*.[5] Moreover, the adversary can dynamically and *indefinitely* delay the response, effectively giving it the power to arbitrarily block honest parties from producing signatures. In ideal-model experiments, the adversary is a well-behaved simulator (which might run the real-world adversary internally, as a subroutine), but in real-world experiments involving protocols that invoke $\mathcal{F}_{\mathsf{sig}}$, the adversary is in general directly under the control of the environment, and this passing of the activation token enables attacks that would not exist if a concrete EUF-CMA signature scheme were used instead.

An astute reader might object that message delivery is also under environmental control in the standard UC model, and therefore progress and termination cannot be guaranteed for interactive protocols in any case. There is, however, a more subtle issue, which enables an adversarial denial-of-service attack to be performed via this formulation of $\mathcal{F}_{\mathsf{sig}}$ even if all messages are delivered and all signature strings are supplied in a timely fashion.

[5]The adversary also sometimes learns when honest parties attempt to verify signatures.

We observe first of all that first-generation $\mathcal{F}_{\mathsf{sig}}$ formulations guarantee unforgeability by forcing verification to fail for all signatures on a message that has never before been signed. Second, we observe that such functionalities guarantee consistency by forcing identical verification queries to always produce the same response. If the UC environment causes a verification request to be performed on a never-signed message and a particular signature string, then this signature string is marked by the functionality as *invalid* for the requested combination of public key and message. If the environment subsequently causes a signing request to be performed on the same message under the same public key, and causes the same signature string to be returned by the adversary, then the functionality is trapped: it must either violate consistency, or else fail to output a signature. In either case, the functionality's behavior diverges from that of any *local* signing algorithm, and generally the authors of prior works have preferred the second option: most first-generation $\mathcal{F}_{\mathsf{sig}}$ formulations specify that the functionality outputs an "error" if this sequence of events occurs.

In this paper, we show that these issues are critical and that in fact the Dolev-Strong protocol *is broken* when instantiated with a first-generation $\mathcal{F}_{\mathsf{sig}}$. Specifically, we recall (in Appendix A) an example of a first-generation signature functionality, and show (in Section 4.5) an attack on the Dolev-Strong protocol instantiated with this $\mathcal{F}_{\mathsf{sig}}$.

The Second Generation of $\mathcal{F}_{\mathsf{sig}}$. The second generation of signature functionalities [Can05, Pat05, KT08, CKKR19] differ from the first in exactly one significant way: rather than interactively querying the adversary in order to obtain public key and signature strings, second-generation functionalities compute these strings internally using algorithms that are supplied by the adversary beforehand. The activation token is never passed to the adversary during honest parties' interactions with the functionality. This eliminates attacks that spring from direct side-channel knowledge of the queries of uncorrupted parties or delayed adversarial responses, but because the string-generation functions are supplied by the adversary,[6] the adversary retains enough power that it can simulate any EUF-CMA signature scheme in the ideal-world experiment. We provide an example of a second-generation functionality for reference in Appendix A.

While this second generation eliminates many simple attacks, the adversary can still render the functionality useless if it fails to provide the string generation algorithms at all. Moreover, the second generation does not make progress toward eliminating the more subtle issue we have described, because the adversary can still supply *bad* algorithms that cause the functionality to emit errors instead of signatures. For example, if the adversary supplies a signature generation algorithm that produces strings that the adversary can predict with noticeable probability, then it can attempt verifications on those strings before they are issued, and force a consistency violation. This is exactly the attack we have described against first-generation functionalities. Because this attack can be performed selectively (e.g., consistency violations can be forced only for certain combinations of message and public key), it also re-opens the door for a more limited form of honest-query leakage. In Section 4.5, we show that the Dolev-Strong protocol is *also* broken when instantiated with a second-generation $\mathcal{F}_{\mathsf{sig}}$.

The Third Generation of $\mathcal{F}_{\mathsf{sig}}$. The third generation of signature functionalities sidestep challenges with adversarially supplied strings and algorithms by *fixing* the exact algorithms in the functionality itself. Examples of such functionalities exist for ECDSA [Lin17, DKLs18, DKLS19, GS22, DKLS24], Schnorr [Lin22], and BBS+ [DKL+23] to name but a few.[7] Third-generation

[6]The adversary, in some sense, uploads a *small piece of itself* to the functionality, which is well-behaved at least with respect to leakage and runtime.

[7]We note that those functionalities were designed for threshold signature MPC protocols. Though the protocols themselves are multiparty, the functionalities they realize are the exact algorithms of the ECDSA, Schnorr, and BBS+

functionalities typically omit any additional checks to ensure that the hard-wired algorithms are well-behaved.[8] The primary caveat with this approach is that hard-wiring the algorithm into the functionality implies that any protocol that *uses* the functionality must make a computational assumption, which implies in turn that the proof of such a protocol cannot be significantly simpler in terms of the reductions required or assumptions employed than the proof of an equivalent protocol that invokes the same algorithms directly, without the functionality.[9]

What We Want in Generation Four. We wish to formulate $\mathcal{F}_{\mathsf{sig}}$ such that it *always* provides perfect signature services to honest parties, regardless of the behavior of the adversary. In other words, a valid signature should always be produced when an honest party makes a signing query, and the adversary should never be able to violate consistency or otherwise render the functionality useless, even with negligible probability. We call such a functionality *unstoppable*. In addition, our functionality should be UC-realized by as many existing signature schemes as possible.

We now give an example to illustrate why *perfect* signature services are useful in a way that signature services with positive but negligible failure probability are not. Suppose that functionality $\mathcal{F}_{\mathsf{sig}}$ has some positive failure probability ν. Consider a higher-level protocol $\pi^{\mathcal{F}_{\mathsf{sig}}}$ that makes queries to $\mathcal{F}_{\mathsf{sig}}$ and fails if $\mathcal{F}_{\mathsf{sig}}$ does. Imagine that there exists some protocol variable x that can assume 2^κ different values, not necessarily according to any particular distribution (perhaps this variable is adversarially influenced, for example), where κ is a security parameter. Let E_i be the event that $x = i$ and the protocol fails (for any reason), and let ε_i be the probability that the protocol fails conditioned on the event that $x = i$. Because we do not know the distribution of x, it would be convenient to use the fact that $\Pr[E_i] \le \varepsilon_i$ to upper bound the probability ε that the protocol fails. Let the overall failure event be E. We have

$$E = \bigcup_{i=1}^{2^\kappa} E_i \qquad \text{and} \qquad \varepsilon \le \sum_{i=1}^{2^\kappa} \varepsilon_i.$$

To shows that ε is negligible in the security parameter, it would be sufficient to upper-bound every ε_i to something very small (e.g., $\varepsilon_i \le 2^{-2\kappa}$). However, if we allow some negligible ν probability that $\mathcal{F}_{\mathsf{sig}}$ fails, it must be the case that every $\varepsilon_i \ge \nu$, and so if $\mathcal{F}_{\mathsf{sig}}$ fails even with probability $\nu = 2^{-\kappa}$, then this bound on ε is useless.

It might be possible to fix this simple example by conditioning on the event that $\mathcal{F}_{\mathsf{sig}}$ does not fail, but this requires a well-defined failure event that captures all deviations of $\mathcal{F}_{\mathsf{sig}}$ from "ideal" behavior, which raises the complexity of this proof pathway considerably. Though we have chosen the union bound as an example, similar problems arise when employing many well-known inequalities from the literature of probabilistic methods over exponentially-sized sets of events. Similar issues would also make a proof by induction much more complicated, if it is feasible at all.

1.3 Our Contributions and Technical Overview

The primary contribution of this work is a new ideal functionality for signatures that is both *unstoppable* and equivalent to exactly the set of all *consistent*, *linear-time* EUF-CMA signatures.[10]

signature schemes. In Section 5 we generalize our result by showing that such functionalities can be used to realize a threshold version of our own functionality.

[8]This implies a fine distinction that we mostly ignore in this work: third-generation functionalities are *signing* functionalities because they enable the act of signing, but they are not necessarily *signature* functionalities because they do not embody the essence of signatures independently of the concrete signature schemes that they run internally.

[9]There may be other advantages of using such a functionality; e.g., structural ones.

[10]We discuss these properties below and formalize them in Section 2.

No matter what the adversary does, it cannot prevent honest parties from obtaining perfectly ideal signature services from our functionality. Our second contribution is a showcase of the usefulness of this unstoppable functionality comprising the first modular analysis of the broadcast protocol of Dolev and Strong [DS83] in the UC framework. Additionally, we show that the Dolev-Strong protocol is broken when instantiated with the first- and second-generations of the ideal signature functionality.

An Unstoppable Signature Functionality. Our functionality must satisfy a set of conflicting requirements. The code of the functionality must be fixed, and yet it must be able to emit the signature strings of a large variety of signature schemes. We resolve this tension in the same way as second-generation functionalities do, by allowing the adversary to upload three algorithms—Gen, Sign, and Verify—that are used to sample public key and signature strings, and verify unsampled ones. However, there is no way for the functionality to test whether the uploaded algorithms are well behaved (this task might even be undecidable). Thus our decision to give the adversary freedom in choosing these algorithms is in tension with our requirement for unstoppability.

We resolve this new tension by imbuing our functionality with an alternate *random* mode: the functionality enters this mode if and only if it detects that the sampling algorithms provided to it have failed or misbehaved (or if they are not provided in a timely fashion), and in this mode the functionality is always guaranteed to provide *perfect* signature services. Although the two modes are easily distinguishable, random mode will never be entered when the adversary supplies a truly EUF-CMA signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$, and our functionality behaves exactly like a second-generation one in that case. Our functionality becomes observably distinct only when a non-EUF-CMA trio of algorithms is supplied.

Random-Mode Signing. We will first explain under what conditions the random mode is triggered, and then we will describe what happens when functionality switches to random mode.

The first time the functionality is activated in some particular session,[11] it immediately requests the algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ from the adversary. If the functionality's second activation in the same session is not accompanied by a response from the adversary (i.e., if the adversary does not respond immediately), then it falls into random mode. If the adversary responds immediately, then in doing so it activates the functionality for the second time in the same session, and the algorithms supplied by the adversary are stored. Thereafter, the functionality enters random mode if and only if the algorithms fail to terminate within a specified time bound[12] or it observes the outputs of $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ to violate a set of invariants that we construe to define useful and well-behaved signatures:

1. The functionality should never respond differently to two identical verification queries. Doing otherwise would violate consistency.

2. The functionality should never emit a signature σ on a message m if it previously returned False to a verification query on the same (m, σ) under the same public key. Doing otherwise would violate correctness or consistency.

3. The functionality should always respond True to a verification query on some message m and signature σ if (m, σ) were previously issued by the functionality under the queried public key.

[11] Sessions are distinguished as usual via a unique session identifier, denoted sid.

[12] We discuss this time bound below.

4. The functionality should never respond True to a verification query on some message m if m was never signed under the queried public key. Doing otherwise would violate unforgeability.

5. The functionality should never emit the same signature in response to signing queries on two different messages under the same public key.

6. The functionality should never emit the same public key twice in response to honest key-generation queries.

Once the functionality enters random mode for some session, it remains in random mode *permanently* with respect to that session. In random mode, the functionality does not use the algorithms provided by the adversary, but instead samples public keys and signatures at random (without replacement), and behaves in such a way as to maintain these invariants. Since there is no a priori bound on the number of keys and signatures that must be generated, and we wish to handle adversaries that can make even an unbounded number of queries, the functionality expands the domain from which keys and signatures are issued, as necessary. Specifically, it maintains length parameters ℓ_{pk} and ℓ_{sig} for those domains, respectively. Both are initialized to 1. When all the keys in $\{0,1\}^{\ell_{\mathsf{pk}}}$ have been issued, the functionality increments ℓ_{pk}; similarly, ℓ_{sig} is incremented when all the signatures in $\{0,1\}^{\ell_{\mathsf{sig}}}$ are used.

Before it switches to random mode, the functionality may have already issued some keys and signatures using the adversarially-provided algorithms. These will continue to be accepted by the verification interface of the functionality (and excluded from the domains keys and signatures that can be sampled) even after the functionality has switched to random mode.

How the Algorithms ($\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}$) are Encoded. Let us now explain how our functionality expects the algorithms ($\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}$) to be encoded. In the literature of signature schemes, Gen is typically modeled as a probabilistic Turing machine: it receives as sole input the security parameter 1^κ in unary, and relies on its random tape to generate keys. The algorithms Sign and Verify are either deterministic or probabilistic.

If the adversary furnishes the algorithms together with the security parameter, and the adversary is free to request signatures on messages of arbitrary length, then there is nothing to prevent the adversary from furnishing algorithms that do not halt within the runtime allotted to the functionality (or within polynomial time, or, indeed, at all). Thus we need an alternative encoding scheme. The most obvious alternative is to require ($\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}$) to be encoded as circuits, which effectively hard-wires the security parameter, and ensures a fixed running time. However, if the signature algorithm is represented as a circuit, then the size of its input is bounded a priori; this is contrary to the typical notion of digital signatures, which accept messages of any length, and it is a significant restriction on the usefulness of the resulting functionality. Therefore, we choose a third approach: we encode all three algorithms as Turing machines and bound them to run in a specific number of steps. For Gen, this number of steps is a constant supplied by the adversary, and for Sign and Verify this number of steps is *linear* in the length of the message, with a coefficient and a constant term supplied by the adversary. The adversary hard-wires the security parameter into these constants and coefficients, and we attain the benefits of the circuit representation without restricting the message space.

It may seem arbitrary to require a signature scheme to have linear running time, but there are three reasons for this particular choice. First, we do not know how to allow the adversary to encode an arbitrary fixed polynomial runtime, without permitting it to encode higher complexity classes. If the adversary were allowed to pick a value c and Sign were allowed $|m|^c$ steps, then the adversary could choose c in $\omega(1)$. Second, if the security parameter is fixed, then practical signature schemes only require linear time (in the message size), since the message is compressed to a fixed size using a hash function that is linear time (again, in the message size). Third, given any signature scheme with polynomial-time Sign and Verify algorithms (in the message size), it is possible to construct another scheme with linear time algorithms: the constructed signing algorithm samples a nonce uniformly from $\mathbb{Z}_{2^\kappa}$, divides the message into chunks of size $\text{poly}(\kappa)$, and for every i signs the i^{th} chunk together with the counter value $i + \mathsf{nonce}$. Verification follows similarly.

Equivalence to EUF-CMA. We prove that our functionality is *equivalent* to a game-based signature definition that comprises four properties. These properties are correctness, existential unforgeability under chosen-message attacks (EUF-CMA), consistency, and linear time. Of these four, only *linear time* is a modeling artifact (as discussed above). EUF-CMA is the usual game-based security definition for signatures. The remaining two properties, correctness and consistency, were discussed in Section 1.1 and deserve further explanation.

Most signature definitions (e.g., [Gol04, KL14]) require perfect correctness, which insists that honestly signed messages *always* verify successfully, and deterministic verification. However, we wish to capture everything that could possibly realize our functionality (and be useful as a signature scheme in the context of Dolev-Strong), and our proof of realization goes through even if both of these properties are violated with negligible probability. Therefore, in Section 2.1 we define a notion *consistent* verification, which insists that Verify always produces the same output when given the same input, except with negligible probability.[13] We also give a weaker correctness definition, which allows for a negligible error. Our weakened correctness definition is distinct from that of Garay et al. [GKZ10]. Though theirs also allows for negligible error, it is too weak to be useful, because they only require correctness for messages chosen before the signing key is sampled. Under this definition, it may not be possible for a signer to sign its own public key. On the other hand, our definition requires correctness for messages generated adaptively by an adversary with knowledge of the public key and all previous signatures.

The equivalence theorem (Theorem 3.2) that we prove in Section 3 takes the following form: First, any signature scheme satisfying the four properties enumerated above UC-realizes our signature functionality when it is embedded in an obvious and trivial protocol.[14] This holds even against malicious adversaries who corrupt parties adaptively. Second, it is possible to extract a signature scheme satisfying our four properties from *any* protocol that realizes our signature functionality in the plain model.

Properties and Capabilities Not Captured. We focus on capturing the core, minimal properties of a signature scheme (i.e., unforgeability, correctness, and consistency), and choose to forgo other capabilities and properties that we view as inessential, even though they may sometimes be useful. For example, our functionality does not provide a means for Alice to transfer her signing powers under a particular public key to Bob, even though she can easily do so by sharing her secret key when a property-based signature definition is used.

We also notably omit from our functionality any notion of identity or any means to establish who "owns" a public key. Parties cannot query the functionality to receive the definitive public key(s) of another party. Such a separation between signing and identity establishment is common in the literature on ideal signatures; functionalities that perform identity establishment are by contrast

[13]Our definition follows that of Garay, Kiayias, and Zhou [GKZ10], in that it applies to adversarially generated public keys. Garay et al. showed a simple separation from the weaker consistency property defined by Canetti [Can04], which only requires Verify to behave consistently for *honestly* generated public keys.

[14]The protocol: parties run the signing algorithms whenever the environment instructs them to, and forward all outputs to the environment, except for secret keys.

often referred to as "authentication," "certification," or "PKI" functionalities (see [Can04, CSV16], for example). While we do make use of a PKI functionality in addition to our signing functionality in our formulation of the Dolev-Strong protocol, we do not enforce "ownership" of established public keys. A corrupt party can pass off an honest party's public key as its own, for example, but if it does so then it can only produce signatures by forging.

Finally, we note that some kinds of signatures have advanced security properties that we have not attempted to model. For example, *unique* signatures are useful for applications such as verifiable random functions, but they require random oracles or strong assumptions to construct, whereas ordinary signatures are feasible assuming only one-way functions. It may be possible to modify our functionality to capture uniqueness, but we leave doing so to future work.

A Modular Analysis of Dolev-Strong Broadcast. As a test case for the usefulness of our signature functionality, we provide a modular analysis of the Dolev-Strong broadcast protocol [DS83] in the UC framework. Together with the UC composition theorem, this result implies that for every MPC protocol in the broadcast model (e.g., the general purpose MPC protocol of [CLOS02] that is UC secure against adaptive corruption of any subset of parties), there exists a protocol with an identical security guarantee that uses only point-to-point communication in the *public key infrastructure* (PKI) model, assuming the existence of EUF-CMA signatures that are linear-time, correct, and consistent.

The Dolev-Strong broadcast protocol assumes a synchronous, fully connected network of point-to-point channels among the participant, and a PKI functionality. It proceeds in a round-by-round manner for $t+1$ rounds, where t is a bound on the number of corrupted parties. Before the protocols begins, the parties use the PKI functionality to reliably distribute their public keys to one another. In the first round, the *sender* signs its message and transmits it to all other parties. Each party P_i maintains a set of values $\mathcal{V}_i$ that is initially empty. In each round ρ, when a party P_i receives a message of the form $(m, \sigma_{j_1}, \ldots, \sigma_{j_{\rho-1}})$ it verifies that it constitutes a *valid signature chain.* That is, it verifies the following conditions:

- σ_{j_1} is a valid signature of the sender on the message m.
- All of the indices $j_1, \ldots, j_{\rho-1}$ are distinct.
- $i \notin \{j_1, \ldots, j_{\rho-1}\}$, i.e., P_i has not yet contributed to this chain.
- For every $k \in \{2, \ldots, \rho-1\}$, the signature σ_{j_k} is a valid signature by P_j on the message $(m, \sigma_{j_1}, \ldots, \sigma_{j_{k-1}})$.

If the signature chain is valid and $m \notin \mathcal{V}_i$, then P_i adds m to $\mathcal{V}_i$, signs the message $(m, \sigma_{j_1}, \ldots, \sigma_{j_{\rho-1}})$ to obtain a signature σ_i, and sends the message $(m, \sigma_{j_1}, \ldots, \sigma_{j_{\rho-1}}, \sigma_i)$ to all parties whose signatures do not appear in the chain. For efficiency reasons, no party need send more than two such chains in the protocol (two chains involving different values of m already provide a proof that the sender has cheated). After round $t+1$, each P_i checks whether $|\mathcal{V}_i| = 1$; if so P_i outputs the value in $\mathcal{V}_i$, and otherwise P_i outputs $\perp$. This concludes the protocol.

It is straightforward to prove that the above protocol achieves a *property-based* definition of broadcast using "ideal signatures" as Dolev and Strong did. *Termination* is guaranteed within $t+1$ rounds. *Validity* (which means that if the sender is honest then all honest parties output its message) follows since every honest party receives the signed message in round 1, and to force an honest party to output $\perp$ the adversary must forge the honest sender's signature. *Agreement* (which requires that all honest parties output the same value, even if the sender cheats) follows from two observations: (1) if an honest party receives a valid signature chain on some message in round $\rho \leq t$, then it is guaranteed that every other honest party will accept this message in round $\rho + 1$, and (2) if an honest party receives a valid signature chain on some message in round $t + 1$, then it is guaranteed that some honest party accepted this message in a prior round, which implies (via our first observation) that all honest parties accept this message in round $t+1$ or earlier. These arguments hold perfectly, by the assumed ideal nature of the signature scheme.

In contrast to the simple proof described above, proving UC security for the Dolev-Strong protocol is a subtle task.

UC is inherently asynchronous, which implies that if we translate the Dolev-Strong protocol naïvely, it is not guaranteed to terminate. Katz et al. [KMTZ13] introduced a method to model synchrony within standard UC, which we adopt. We refer the reader to Section 4.1 and Appendix B for an overview of this model, and give here a short account of its relevant features. The method of Katz et al. augments the protocol with an ideal clock functionality to synchronize honest parties and with bounded-delay channels. Indistinguishability of the real and ideal worlds can only be proven if the *round structure* of the real and ideal computations is the same. Therefore, the ideal broadcast functionality must keep a local round counter that advances based on activations it gets from honest parties, in a way that mimics the round counter in the real protocol. Our formalism follows that of Cohen et al. [CCGZ16] in that it separates the ideal functionality into two components: a simple *canonical synchronous functionality* (CSF) that has an input round and an output round and contains the core of the functionality's code, and a protocol-dependent *round-extending wrapper* that is responsible for maintaining the logistics of the round counter, etc. This approach allows our proof to be adjusted to accommodate alternative broadcast protocols such as the Katz-Koo protocol [KK06] and the protocol of Garay et al. [GKKO07] which have expected-constant round counts and probabilistic termination.

In the spirit of [CR03], we wish to capture multiple instances of broadcast that share a single PKI. Correspondingly, each session of our broadcast functionality comprises a single setup phase and multiple subsequent broadcast phases that are distinguished by unique sub-session IDs that specify the sender and a specific subset of recipients. Such unique IDs are known to be necessary in order to support composition [LLR02]. Our broadcast functionality is formally specified in Section 4.2.

In Section 4.3 we reformulate Dolev-Strong modularly, in the $(\mathcal{F}_{\mathsf{sig}}, \mathcal{F}_{\mathsf{pki}})$-hybrid model. Much like our ideal broadcast functionality, the protocol consists of multiple phases. In the setup phase, each party obtains a public key from the ideal signature functionality $\mathcal{F}_{\mathsf{sig}}$ and registers it with the other parties via the PKI functionality $\mathcal{F}_{\mathsf{pki}}$. In subsequent broadcast phases (which might be concurrent), the parties execute the Dolev-Strong protocol as described above, using $\mathcal{F}_{\mathsf{sig}}$ to furnish the signatures.

The simulator we devise for Dolev-Strong (in Section 4.4) is relatively straightforward, which we view as evidence that our signature and broadcast functionalities are well-formulated. Nevertheless, there are a few subtleties. The simulator cannot send a corrupted sender's input to the broadcast functionality in the first round, since the adversary might send a conflicting signed message later. Furthermore, the simulator cannot be sure that honest parties should output $\perp$ even if some honest party receives two conflicting signed messages at some point in the simulation, because this party might later be corrupted adaptively. As a result, the simulator must wait until the final round of the protocol to send its input to the broadcast functionality. We emphasize that such issues as these *do not* arise when proving that a protocol achieves a property-based notion of broadcast, but they are crucial in simulation-based proofs.

Finally, we do not bound the number of corruptions by t and the number of rounds by $t+1$ as the original proof of Dolev-Strong does. Instead, we run the protocol for as many rounds as there are parties, and consider a setting where any subset of parties might be adaptively corrupted (including, potentially, *all* of them). In the standalone setting under static corruptions, a completely corrupt protocol is a trivial case, but in the UC model under adaptive corruptions, this case is known to be extremely challenging. However, including it in our proof allows our UC version of Dolev-Strong to be used in the context of adaptively secure MPC protocols (e.g., [CLOS02]).

Generalizing to the Threshold Setting. In Section 5 we prove that our result generalizes to the threshold setting. Specifically, we show that we can realize a threshold variant of $\mathcal{F}_{\mathsf{sig}}$ given a functionality that computes the functions of a consistent, linear-time EUF-CMA signature scheme in a threshold fashion.[15] In other words, we propose the first threshold ideal signature with *truly* guaranteed output, and demonstrate that it can be realized.

2 Preliminaries

Notation. We use $=$ to denote equality, $:=$ for right-to-left assignment, and $\leftarrow$ for sampling from a distribution (often defined by a probabilistic algorithm) or uniformly from a set. We use bracket notation to generate inclusive ranges, so $[n]$ denotes the integers from 1 to n and $[5, 7] = \{5, 6, 7\}$. In general, single-letter variables are set in *italic* font, multi-letter variables and function names are set in $\mathsf{sans\text{-}serif}$ font, string literals are set in $\texttt{slab} - \texttt{serif}$ font, and we use **bold** to denote vectors.

2.1 Digital Signature Schemes

Definition 2.1 (Digital Signature Scheme [GMR88, Gol04, KL14])**.** *A* signature scheme *is a tuple of probabilistic polynomial-time (PPT) algorithms,* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *such that:*

1. *Given a security parameter* κ*, the* Gen *algorithm outputs a public key/secret key pair:* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa)$*.*

2. *Given a security parameter* κ*, a secret key* sk*, and a message* m*, the* Sign *algorithm outputs a signature* σ*:* $\sigma \leftarrow \mathsf{Sign}(1^\kappa, \mathsf{sk}, m)$*.*

3. *Given a security parameter* κ*, a message* m*, a signature* σ*, and a public key* pk*, the* Verify *algorithm outputs a bit* b *indicating whether the signature is valid or invalid:* $b \leftarrow \mathsf{Verify}(1^\kappa, \mathsf{pk}, m, \sigma)$*.*

Digital Signatures are almost always required to fulfill the properties of *correctness* and *existential unforgeability* (EUF-CMA). In addition to these two, we add the properties of *consistency* and *linear-timeness.*

We begin with correctness. The standard form the correctness property (e.g., [Gol04, KL14]) insists that it be perfect, and a few works have used a relaxed definition that permits negligible correctness error (e.g., [Can04, GKZ10]). We also permit negligible error, but strengthen the property by allowing the adversary to see the public verification key and signatures on messages of its choice during its attempts to cause a correctness error.

Definition 2.2 (Correctness)**.** *A signature scheme* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is* correct *if for all PPT adversaries* $\mathcal{G}$*, the following experiment outputs* 1 *with probability negligible in* κ*.*

[15] As we discussed in Section 1.2, such functionalities have been proposed by many prior works on threshold signing; e.g., see [Lin17, DKLs18, DKLS19, Lin22, GS22, DKL+23, DKLS24].

1. *Generate the keys* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathit{Gen}(1^\kappa)$.

2. *Give* pk *to the adversary* $\mathcal{G}$*, who generates a stream of messages to sign. For each message* m*, run* $\sigma \leftarrow \mathit{Sign}(1^\kappa, \mathsf{sk}, m)$ *and* $b \leftarrow \mathit{Verify}(1^\kappa, \mathsf{pk}, m, \sigma)$*. If* $b = 0$*, exit the experiment with output* 1*. If* $b = 1$*, return* σ *to* $\mathcal{G}$*.*

3. *When* $\mathcal{G}$ *finishes, output* 0*.*

Whereas our correctness definition differs slightly from prior works, our existential unforgeability definition is completely standard.

Definition 2.3 (EUF-CMA [KL14])**.** *A signature scheme* $\Sigma = (\mathit{Gen}, \mathit{Sign}, \mathit{Verify})$ *is* EUF-CMA *if for all PPT adversaries* $\mathcal{G}$*, the following experiment outputs* 1 *with probability negligible in* κ*.*

1. *Generate the keys* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathit{Gen}(1^\kappa)$.

2. *Adversary* $\mathcal{G}$ *is given* pk *and oracle access to* $\mathit{Sign}(1^\kappa, \mathsf{sk}, \cdot)$*. The adversary then outputs* (m, σ)*.*

3. *Let* $\mathcal{Q}$ *denote the set of messages whose signatures were requested from the signing oracle by* $\mathcal{G}$ *during its execution. The output of the experiment is* 1 *if* $m \notin \mathcal{Q}$ *and* $\mathit{Verify}(1^\kappa, pk, m, \sigma) = 1$*.*

Most prior signature functionalities require that Verify be deterministic, while we generalize the signature scheme to allow Verify to be probabilistic.[16] However, probabilistic verification brings with it the possibility of disagreement upon the validity of a signature, which can be fatal for consensus protocols such as Dolev-Strong. We therefore define a *consistency* property, which requires that on every input, Verify must either output 0 with all but negligible probability, or output 1 with all but negligible probability. That is, the probability that Verify returns different outputs when run twice on the same input must be negligible. A similar consistency requirement was previously defined by Garay, Kiayias, and Zhou [GKZ10], who also showed it to be stronger than Canetti's consistency property [Can04], which only holds for honestly generated public keys.

Definition 2.4 (Consistency)**.** *A signature scheme* $\Sigma = (\mathit{Gen}, \mathit{Sign}, \mathit{Verify})$ *is* consistent *if for all PPT adversaries* $\mathcal{G}$ *the following experiment outputs* 1 *with probability negligible in* κ*.*

1. *The adversary samples an arbitrary public key, message, and signature:* $(\mathsf{pk}, m, \sigma) \leftarrow \mathcal{G}(1^\kappa)$*.*

2. *The experiment samples two independent appropriate-length random bit-strings,* r_1 *and* r_2*.*

3. *The experiment outputs* 1 *if* $\mathit{Verify}(1^\kappa, \mathsf{pk}, m, \sigma; r_1) \neq \mathit{Verify}(1^\kappa, \mathsf{pk}, m, \sigma; r_2)$

Finally, we introduce a novel property, which insists on a stricter time bound on the signature algorithms than the typical simple "polynomial time" requirement. Specifically, while runtime may grow polynomially with the security parameter, we insist that it grow at most linearly with the message length. Looking ahead, this is necessary because the adversary will supply arbitrary signature algorithms to our functionality, and the functionality must run them within a similarly strict time bound that is enforced by the UC model. In other words, this property is essentially a modeling artifact. We refer the reader to Section 1.3 for further discussion.

Definition 2.5 (Linear Time)**.** *A signature scheme* $\Sigma = (\mathit{Gen}, \mathit{Sign}, \mathit{Verify})$ *is* linear time *if there exists a polynomial* $T(\kappa)$ *such that* $\mathit{Sign}(1^\kappa, \mathsf{sk}, m)$ *and* $\mathit{Verify}(1^\kappa, \mathsf{pk}, m, \sigma)$ *always halt within* $T(\kappa) \cdot (\ell + 1)$ *steps, where* ℓ *is the total length of the input.*

[16]This decision is partially motivated by the fact that we wish to prove that a signature scheme can always be extracted from any protocol that realizes our functionality, and we cannot rule out probabilistic verification in the extracted scheme.

We note that any signature scheme that does not satsify the linear-time requirement can be converted generically into a signature scheme that does, if the signature size is permitted to grow linearly with the message size. The new signing algorithm first partitions the message into chunks of size κ, and then uses the original signing algorithm to sign the first chunk. Each *subsequent* chunk is concatenated with the signature of its predecessor, and then the concatenation is signed with the original signing algorithm. The final signature is the set of intermediate chunk-signatures. If linear-time collision resistant hash functions are permitted, then the conversion is simpler still: the message is simply hashed before signing.

2.2 The Universal Composability Framework

In this section, we given a high-level (and somewhat informal) description of the UC framework. We direct the reader to Canetti for further details [Can20].

The Real Model. An execution of a protocol π in the real model consists of n PPT *interactive Turing machines* (ITMs) $P_1, \ldots, P_n$ representing the parties, along with two additional ITMs: an *adversary* $\mathcal{A}$, describing the behavior of the corrupted parties and an *environment* $\mathcal{Z}$, representing the external network environment in which the protocol operates. The environment gives inputs to the honest parties, receives their outputs, and can communicate with the adversary at any point during the execution. The adversary exercises complete control over the corrupted parties and may cause them to act in arbitrary ways.

Each ITM in the experiment is initialized with the security parameter κ and a tape that supplies random coins, and the environment receives an additional auxiliary input tape. At every point during the experiment, exactly one ITM is actively running, and an ITM can *activate* another one to which it is connected by a tape, granting the target ITM the ablity to run, while relinquishing its own privilege to do so. The environment is activated first. When active, it can read the output tapes of all honest parties and of the adversary, and it can activate one of the parties or the adversary by writing on their input tapes. When a party is active, it can perform a local computation, write on its output tape, or send messages to other parties by writing on its outgoing communication tapes. After a party completes its operations, control is returned to the environment. When the adversary is activated it can send messages on behalf of the corrupted parties, or send a message to the environment by writing on its output tape. $\mathcal{A}$ also controls the communication between the parties by reading the contents of the messages on the outgoing communication tapes of honest parties and writing messages on their incoming communication tapes. In addition, $\mathcal{A}$ can corrupt an honest party, at which point it gains complete read and write access to its tapes and the state of its internal memory. Whenever a party is corrupted the environment is notified. When $\mathcal{A}$ completes its operations, if it wrote on the incoming tape of an honest party, then that party is activated next, and the environment is activated otherwise. The protocol is complete when $\mathcal{Z}$ outputs a single bit.

Let $\textsc{Real}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, z; \boldsymbol{r})$ denote $\mathcal{Z}$'s output on input z and security parameter κ, after interacting with adversary $\mathcal{A}$ and parties $P_1, \ldots, P_n$ who run protocol π with random tapes $\boldsymbol{r} = (r_1, \ldots, r_n, r_{\mathcal{A}}, r_{\mathcal{Z}})$ as described above. Let $\textsc{Real}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, z)$ denote the random variable $\textsc{Real}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, z; \boldsymbol{r})$ when the vector $\boldsymbol{r}$ is uniformly chosen.

The Ideal Model. A computation in the ideal model consists of n *dummy* parties $P_1, \ldots, P_n$, an *ideal-process adversary* (otherwise known as a *simulator*) $\mathcal{S}$, an environment $\mathcal{Z}$, and an *ideal functionality* $\mathcal{F}$. As in the real model, the environment gives inputs to the honest (dummy) parties, receives their outputs, and communicates with the simulator at any point during the execution.

The dummy parties act as channels between the environment and the ideal functionality, simply forwarding any messages receveied from $\mathcal{Z}$ to $\mathcal{F}$, and vice versa. The ideal functionality $\mathcal{F}$ defines the desired behavior of the computation. $\mathcal{F}$ receives inputs (and activations) from the dummy parties, executes the desired computation and sends the output to the parties. The ideal-process adversary does *not* observe or mediate the communication between the parties and the ideal functionality; however, $\mathcal{S}$ can communicate with $\mathcal{F}$ *directly* via the *back-door tape* of $\mathcal{F}$, if the specification of $\mathcal{F}$ permits such communcation.

Let $\textsc{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, z; \boldsymbol{r})$ denote $\mathcal{Z}$'s output on input z and security parameter κ, after interacting with ideal-process adversary $\mathcal{S}$ and dummy parties $P_1, \ldots, P_n$ that interact with ideal functionality $\mathcal{F}$ with random tapes $\boldsymbol{r} = (r_{\mathcal{F}}, r_{\mathcal{S}}, r_{\mathcal{Z}})$ as described above. Let $\textsc{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, z)$ denote the random variable $\textsc{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, z; \boldsymbol{r})$ when the vector $\boldsymbol{r}$ is uniformly chosen.

Definition 2.6. *We say that a protocol π* UC-realizes *an ideal functionality $\mathcal{F}$ in the presence of adaptive malicious*[17] *adversaries, if for every PPT adaptive malicious adversary $\mathcal{A}$ there exists a PPT ideal-process adversary $\mathcal{S}$ such that for every PPT environment $\mathcal{Z}$, the following two distribution ensembles are computationally indistinguishable*

$$\{\textsc{Real}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, z)\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*} \approx_{\mathrm{c}} \{\textsc{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, z)\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}\,.$$

The Hybrid Model. The *$\mathcal{F}$-hybrid model* is a combination of the real and ideal models; it extends the real model with an ideal functionality $\mathcal{F}$. The parties communicate with each other in exactly the same way as in the real model described above; however, they can interact with (and activate) $\mathcal{F}$ as in the ideal model, and $\mathcal{F}$ can communicate with $\mathcal{A}$ via the back-door tape. Hybrid models can be defined with respect to multiple functionalities, and protocols defined with respect to these hybrid models (i.e., protocols can involve functionalities in addition to real parties). The main useful property and raison d'être of the UC framework is its support for a composition operation: the ideal functionality $\mathcal{F}$ in an $\mathcal{F}$-hybrid model can be replaced with a protocol that UC-realizes $\mathcal{F}$, with negligible security loss, regardless of how $\mathcal{F}$ is invoked. This guarantee is formalized in Theorem 2.7.

Let the global output $\textsc{Hybrid}^{\mathcal{F}}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, z)$ denote $\mathcal{Z}$'s output on input z and security parameter κ, after interacting in a $\mathcal{F}$-hybrid model with adversary $\mathcal{A}$ and parties $P_1, \ldots, P_n$ who run protocol π with uniformly distributed random tapes.

Theorem 2.7 ([Can20])**.** *Let $\mathcal{F}$ be an ideal functionality and let ρ be a protocol that UC-realizes $\mathcal{F}$ in the presence of adaptive malicious adversaries, and let π be a protocol in the $\mathcal{F}$-hybrid model, and π^ρ be a second protocol identical to π, except that whereever a party activates $\mathcal{F}$ in π, it instead invokes the matching interface of ρ in π^ρ. For every PPT adaptive malicious real-model adversary $\mathcal{A}$ there exists a PPT adaptive malicious adversary $\mathcal{S}$ in the $\mathcal{F}$-hybrid model such that for every PPT environment $\mathcal{Z}$, it holds that*

$$\left\{\textsc{Real}_{\pi^\rho,\mathcal{A},\mathcal{Z}}(\kappa, z)\right\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*} \approx_{\mathrm{c}} \left\{\textsc{Hybrid}^{\mathcal{F}}_{\pi,\mathcal{S},\mathcal{Z}}(\kappa, z)\right\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}.$$

Relevant Limitations of the UC Framework. We note that because the adversary controls communication between honest parties, the adversary can prevent any protocol in the plain UC framework from terminating. In other words, UC does not capture the notion of *Guaranteed Output Delivery*, which is a key aspect of broadcast and Byzantine agreement without which both

[17]Here *adaptive* refers to the fact that the adversary can corrupt honest parties at any time during the experiement, and *malicious* refers to the fact that corrupted parties might behave arbitrarily. Weaker corruption models exist, but we are not concerned with them in this work.

problems become very easy to solve. Furthermore, the plain UC model is inherently asynchronous and provides no means to guarantee synchronicity, as required to express Dolev-Strong, for example. Follow-up works have addressed these shortcomings without altering the basic framework, by building additional abstractions atop UC. We explore these follow-up works in Section 4.1.

3 An Unstoppable Signature Functionality

In this section, we present our signature functionality $\mathcal{F}_{\mathsf{sig}}$. Section 3.1 contains the formal specification, and Section 3.2 a proof of equivalence to consistent linear-time EUF-CMA signatures.

3.1 The Functionality

For an overview of our functionality and a discussion of our design choices (such as random mode and the length parameters ℓ_{pk} and ℓ_{sig}), we direct the reader to Section 1.3. We wish to make two further remarks.

On Necessity of Per-Party Initialization. In Section 1.3, we specified that the adversary supplies a trio of algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ to the functionality, but we did not specify a mechanism by which it should do so. This is a non-trivial concern. There is no guarantee that the adversary (ideal or otherwise) will be activated *before* the honest parties first query the functionality. If the functionality is first activated by an honest party that expects a *response*, and then activates the adversary to retrieve the algorithms, the adversary has an opportunity to pass activation to the environment (or some other party) instead of replying to the functionality, again denying a timely response to the requesting party. Worse, when activation passes back to the requesting party, there will be no response from the functionality on its subroutine-output tape, and it is not clear in general how the party will react to this situation (the UC framework does not specify, so far as we are aware); the experiment may even deadlock.

To resolve this difficulty, we insist that the functionality must be activated at least once *without* the expectation of a response before it can be accessed *with* the expectation of a response. If a response-expecting activation occurs before the adversary suppies the algorithms, then by definition they have not been delivered in a timely fashion, and the functionality switches into random mode. To facilitate this, we add to our functionality an *initialization* interface that produces no response; since we do not wish our functionality to imply communication between parties, we insist that each party invoke this interface independently before that party invokes any other interfaces. This new interface does not have an analogue in prior works.

On The Precise Syntax and Runtime Bounds of the $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ Algorithms. In Section 1.3, we argued that we do not know how to allow the adversary to encode an arbitrary fixed polynomial runtime, without permitting it to encode higher complexity classes. Nevertheless, we wish to allow allow the algorithms time polynomial in the security parameter κ. Fortunately, κ is fixed per session, unlike the message length. Thus we can insist that the adversary *hardcode* κ into the algorithms it supplies.[18] We next interpret the combined length of the algorithms supplied as an upper bound on the number of computational steps they each require per bit of the message. This is equivalent to forcing the adversary to supply an explicit upper bound in unary; the adversary can give its algorithms more time simply by padding their length with no-op instructions.

[18]If the adversary hardcodes the wrong value, this is treated just like any other corrupt algorithm.

Functionality 3.1. $\mathcal{F}_{\mathsf{sig}}$ (An Unstoppable Signature Functionality)

This functionality interacts with an ideal adversary $\mathcal{S}$ and a number of real parties (all of them denoted P) that is not a-priori known. For simplicity of description, we assume this functionality has *per-session* memory. That is, all stored and recalled values are associated with the particular session ID sid of the query that generated them. Note that P may refer to a different party in every interaction.

Initialization.

1. Ignore any message from any party P that contains some session ID sid until *after* party P sends $(\mathtt{init}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{sig}}$.
2. Upon receiving $(\mathtt{init}, \mathsf{sid})$ for the *first time* for some particular sid, send $(\mathtt{init}, \mathsf{sid})$ to $\mathcal{S}$ and wait.
3. Upon receiving any second message that contains the session ID sid after the first $(\mathtt{init}, \mathsf{sid})$ message (regardless of whether the same party transmitted the two messages):
   (a) If the message arrived from $\mathcal{S}$ and is of the form $(\mathtt{algs}, \mathsf{sid}, \Sigma)$ where $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}) := \Sigma$ is the description of three probabilistic Turing machines, store $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ and $s := |\Sigma|$ in memory and set the flag $\mathsf{rmode} := 0$.
   (b) Otherwise, set the flag $\mathsf{rmode} := 1$.

   Regardless, set the integers $\ell_{\mathsf{pk}} := 1$ and $\ell_{\mathsf{sig}} := 1$, and initialize the set of assigned public keys $\mathcal{K} := \emptyset$ and the set of assigned signatures $\mathcal{Q} := \emptyset$. If $\mathsf{rmode} = 1$, process the second message for sid using the interfaces below.

Key Generation.

4. Upon receiving $(\mathtt{keygen}, \mathsf{sid})$ from a party P,
   (a) If $\mathsf{rmode} = 0$, then sample a uniformly random bit-string r_k of appropriate length,[a] and compute $(\mathsf{sk}, \mathsf{pk}) := \mathsf{Gen}(r_k)$. If $\mathsf{pk} \in \mathcal{K}$ or Gen does not terminate in s computational steps, then switch to random mode by setting $\mathsf{rmode} := 1$ and following the instruction below for the case that $\mathsf{rmode} = 1$.
   (b) If $\mathsf{rmode} = 1$, then sample $\mathsf{pk} \leftarrow \{0,1\}^{\ell_{\mathsf{pk}}} \setminus \mathcal{K}$ uniformly and set $\mathsf{sk} := \bot$ and $r_k := \bot$.

   Regardless, update $\mathcal{K} := \mathcal{K} \cup \{\mathsf{pk}\}$ in memory and increment ℓ_{pk} until $\{0,1\}^{\ell_{\mathsf{pk}}} \setminus \mathcal{K} \neq \emptyset$. Store $(\mathtt{key}, \mathsf{sid}, P, \mathsf{pk}, \mathsf{sk}, r_k)$ in memory and send $(\mathtt{public\text{-}key}, \mathsf{sid}, \mathsf{pk})$ to the caller P.

Signing.

5. Upon receiving $(\mathtt{sign}, \mathsf{sid}, \mathsf{pk}, m)$ from a party P, update $\mathcal{K} := \mathcal{K} \cup \{\mathsf{pk}\}$, and increment ℓ_{pk} until $\{0,1\}^{\ell_{\mathsf{pk}}} \setminus \mathcal{K} \neq \emptyset$. Check if a record of the form $(\mathtt{key}, \mathsf{sid}, P, \mathsf{pk}, \mathsf{sk}, r_k)$ exists in memory for any $\mathsf{sk} \in \{0,1\}^* \cup \{\bot\}$ and any r_k. If not, return $\bot$ to P. Otherwise:
   (a) If $\mathsf{rmode} = 0$, then sample a uniformly random bit-string r_σ of appropriate length,[a] compute $\sigma := \mathsf{Sign}(\mathsf{sk}, m; r_\sigma)$ and check the following conditions:
   - $(\mathtt{sig}, \mathsf{sid}, \mathsf{pk}, m', \sigma, r_\sigma)$ exists in memory such that $m \neq m'$.
   - $(\mathtt{bad\text{-}sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ exists in memory.

- Sign does not terminate in $(|m|+1) \cdot s$ computational steps.

  If any of the above conditions holds, then switch to random mode by setting $\mathsf{rmode} \coloneqq 1$ and following the instruction below for the case that $\mathsf{rmode} = 1$.

(b) If $\mathsf{rmode} = 1$, then sample $\sigma \leftarrow \{0,1\}^{\ell_{\mathsf{sig}}} \setminus \mathcal{Q}$ and set $r_\sigma \coloneqq \bot$.

Regardless, update $\mathcal{Q} \coloneqq \mathcal{Q} \cup \{\sigma\}$ and increment ℓ_{sig} until $\{0,1\}^{\ell_{\mathsf{sig}}} \setminus \mathcal{Q} \neq \emptyset$. Store $(\texttt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, r_\sigma)$ in memory and return $(\texttt{signature}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ to the caller P.

Verification.

6. Upon receiving $(\texttt{verify}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ from some party P, update $\mathcal{K} \coloneqq \mathcal{K} \cup \{\mathsf{pk}\}$, and increment ℓ_{pk} until $\{0,1\}^{\ell_{\mathsf{pk}}} \setminus \mathcal{K} \neq \emptyset$. Next, scan the memory for records of the form $(\texttt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, *)$ or $(\texttt{bad-sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$, for any σ, and for a record of the form $(\texttt{key}, \mathsf{sid}, P', \mathsf{pk}, *, *)$ for any P'.[b]

   (a) If the $\texttt{sig}$ record exists, then set $b \coloneqq 1$.

   (b) If there is no $\texttt{sig}$ record, but there is a $\texttt{key}$ record and P' is an honest party, then set $b \coloneqq 0$.

   (c) If there is no $\texttt{sig}$ record, but the $\texttt{bad-sig}$ record exists, then set $b \coloneqq 0$.

   (d) If Steps 6a through 6c do not apply, and $\mathsf{rmode} = 1$, then set $b \coloneqq 0$.

   (e) If Steps 6a through 6c do not apply, and $\mathsf{rmode} = 0$, then set $b \leftarrow \mathsf{Verify}(\mathsf{pk}, m, \sigma)$. If Verify does not produce output before $(|m|+1) \cdot s$ computational steps have elapsed, then terminate its execution, set $b \coloneqq 0$, and switch to random mode by setting $\mathsf{rmode} \coloneqq 1$ in memory.

   If, after evaluating the above conditions, $b = 0$ but the record $(\texttt{bad-sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ is not stored in memory, then store it.

   If, after evaluating the above conditions, $b = 1$ but no record of the form $(\texttt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, *)$ exists in memory, then store $(\texttt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, \bot)$.

   Regardless, update $\mathcal{Q} \coloneqq \mathcal{Q} \cup \{\sigma\}$ in memory and increment ℓ_{sig} until $\{0,1\}^{\ell_{\mathsf{sig}}} \setminus \mathcal{Q} \neq \emptyset$. Finally, return $(\texttt{verified}, \mathsf{sid}, \mathsf{pk}, m, \sigma, b)$ to P.

Corruption.

7. Upon receiving $(\texttt{corrupt}, \mathsf{sid}, P)$ from $\mathcal{S}$, search the memory for all records of the form $(\texttt{key}, \mathsf{sid}, P, \mathsf{pk}, \mathsf{sk}, r_k)$, and for each such record compute the set $\mathcal{C}_{\mathsf{pk}}$ of all (m, σ, r_σ) such that there exists a record of the form $(\texttt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, r_\sigma)$ in memory. Return $(\texttt{corrupt}, \mathsf{sid}, P, \mathcal{C})$ to $\mathcal{S}$, where $\mathcal{C}$ is a set containing $(\mathsf{pk}, \mathsf{sk}, r_k, \mathcal{C}_{\mathsf{pk}})$ for every $(\texttt{key}, \mathsf{sid}, P, \mathsf{pk}, \mathsf{sk}, r_k)$ that was found.

[a] We assume that the amount of randomness that Gen, Sign, and Verify need is part of their description.

[b] P' may or may not be the same as P.

3.2 Equivalence to Consistent Linear-Time EUF-CMA

We prove two main theorems about $\mathcal{F}_{\mathsf{sig}}$. First, in this section, we prove Theorem 3.2, which assumes the existence of a simple non-interactive protocol that calls three algorithms that have the syntax of a signature scheme, and shows that if the algorithms achieve the game-based security notion of signatures, then the protocol realizes our functionality, and vice versa. Then, in Section 3.3, we prove Theorem 3.7, which generalizes the aforementioned equivalence beyond protocols that are explicitly structured like signatures by demonstrating that it is possible to *extract* a secure game-based signature from *any* protocol that realizes $\mathcal{F}_{\mathsf{sig}}$ in the plain model.

A Real-World Dummy Protocol. As in the prior works of Canetti [Can01, Can04], we specify a simple *dummy protocol* π_{Σ} that allows us to straightforwardly convert a signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ into a real-world UC experiment that is a syntactic match for $\mathcal{F}_{\mathsf{sig}}$.[19] Each party ignores all instructions from the environment in the session sid until after it receives $(\mathtt{init}, \mathsf{sid})$. Thereafter, if a party receives $(\mathtt{keygen}, \mathsf{sid})$, it runs $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\kappa)$, stores sk, and outputs $(\mathtt{public\text{-}key}, \mathsf{sid}, \mathsf{pk})$. When P subsequently receives $(\mathtt{sign}, \mathsf{sid}, \mathsf{pk}, m)$, it computes $\sigma \leftarrow \mathsf{Sign}(1^\kappa, \mathsf{sk}, m)$ and outputs $(\mathtt{signature}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$. If any party receives $(\mathtt{verify}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$, it sets $b \leftarrow \mathsf{Verify}(1^\kappa, \mathsf{pk}, m, \sigma)$ and returns $(\mathtt{verified}, \mathsf{sid}, \mathsf{pk}, m, \sigma, b)$ to the environment.

Theorem 3.2. *Let Σ be a linear-time signature scheme (Definition 2.5). The following are equivalent:*

(1) Σ is correct (Definition 2.2), EUF-CMA secure (Definition 2.3), and consistent (Definition 2.4).

(2) π_{Σ} UC-realizes $\mathcal{F}_{\mathsf{sig}}$ with security against a malicious adversary who statically corrupts any number of parties.

(3) π_{Σ} UC-realizes $\mathcal{F}_{\mathsf{sig}}$ with security against a malicious adversary who adaptively corrupts any number of parties.

Proof. Trivially, we have (3) $\implies$ (2). The proof then follows from Lemmas 3.5 and 3.6, which show that (1) $\implies$ (3), and (2) $\implies$ (1), respectively. □

We begin with two intermediate lemmas which will be necessary for our proof of Lemma 3.5.

Lemma 3.3. *If $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ is a correct, EUF-CMA secure signature scheme, then for any polynomial $B(\kappa)$ and PPT adversary $\mathcal{G}$ outputting a set $\mathcal{K}$ of size at most $B(\kappa)$, there exists a negligible function* negl *such that*

$$\Pr[\mathsf{pk} \in \mathcal{K} : \mathcal{K} \leftarrow \mathcal{G}(1^\kappa), (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\kappa)] \le \mathrm{negl}(\kappa).$$

Proof. Let $p_0(\kappa)$ be the probability that $\mathsf{pk} \in \mathcal{K}$ in the experiment defined in Lemma 3.3. We wish to bound this value from above. First, consider a modified experiment wherein, if $\mathsf{pk} \in \mathcal{K}$, the challenger generates keys $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{Gen}(1^\kappa)$ for $i \in [\lfloor B(\kappa) \rfloor]$, and checks whether $\mathsf{pk} = \mathsf{pk}_i$ for some i. Let $p_1(\kappa)$ be the probability that $\mathsf{pk} \in \mathcal{K}$ *and* $\mathsf{pk} = \mathsf{pk}_i$ for at least one i. We have

$$p_1(\kappa) \ge \frac{p_0(\kappa) \cdot (1 - e^{-1}) \cdot \min(1, B(\kappa) \cdot p_0(\kappa))}{|\mathcal{K}|} \ge (1 - e^{-1}) \cdot \min\Big(\frac{p_0(\kappa)}{|\mathcal{K}|}, p_0(\kappa)^2\Big),$$

because the probability that at least one of the $B(\kappa)$ independent samples pk_i is in $\mathcal{K}$ is $1 - (1 - p_0(\kappa))^{B(\kappa)} \ge 1 - e^{-B(\kappa) \cdot p_0(\kappa)} \ge (1 - e^{-1}) \cdot \min(1, B(\kappa) \cdot p_0(\kappa))$, and the probability that two independent samples from a distribution supported on $\mathcal{K}$ are equal is at least $|\mathcal{K}|^{-1}$.[20]

[19] This is a basic precondition for any proof that the former UC-realizes the latter.

[20] One could generate more pk_i values to increase this towards $(1 - e^{-1}) \cdot p_0(\kappa)/|\mathcal{K}|$, at the cost of computation proportional to $1/p_0(\kappa)$. This would still leave the expected computation essentially the same as $\mathcal{G}$, however, as this computation only occurs when $\mathsf{pk} \in \mathcal{K}$, i.e., with probability $p_0(\kappa)$.

Next, consider a further-modified experiment that also generates a signature $\sigma \leftarrow \mathsf{Sign}(1^\kappa, \mathsf{sk}, 0)$. Let $p_2(\kappa)$ be the probability that the conditions for p_1 and p_0 hold *and* $\mathsf{Verify}(1^\kappa, \mathsf{pk}, 0, \sigma)$ outputs 1. By reduction to the correctness of Σ, we have $p_2(\kappa) \geq p_1(\kappa) - \mathrm{negl}'(\kappa)$ for some negligible function $\mathrm{negl}'(\kappa)$.

Now consider one final modified experiment that generates σ using sk_i instead of sk, where i is the lowest index for which $\mathsf{pk}_i = \mathsf{pk}$. The probability $p_2(\kappa)$ is unchanged in this experiment, relative to the experiment in which it was defined: for every sequence of key pairs $((\mathsf{sk}, \mathsf{pk}), (\mathsf{sk}_1, \mathsf{pk}_1), \ldots, (\mathsf{sk}_i, \mathsf{pk}_i), \ldots, (\mathsf{sk}_{B(\kappa)}, \mathsf{sk}_{B(\kappa)}))$ such that i is the first index for which $\mathsf{pk} = \mathsf{pk}_i$, the probability that this sequence is sampled by $\mathsf{Gen}(1^\kappa)$ is exactly the same as the probability of sampling the sequence $((\mathsf{sk}_i, \mathsf{pk}), (\mathsf{sk}_1, \mathsf{pk}_1), \ldots, (\mathsf{sk}, \mathsf{pk}_i), \ldots, (\mathsf{sk}_{B(\kappa)}, \mathsf{sk}_{B(\kappa)}))$, wherein sk and sk_i have been swapped.

Finally, we can bound $p_2(\kappa)$ in the last experiment via reduction to the EUF-CMA security of Σ. Let pk be the challenge instance; the reduction continues the experiment until it gets $(\mathsf{sk}_i, \mathsf{pk}_i)$ (with $\mathsf{pk}_i = \mathsf{pk}$), generates the signature σ using sk_i, and outputs $(0, \sigma)$. The reduction has made no signing queries, which implies that the EUF-CMA experiment will output 1 if σ is correct. Therefore, the reduction has advantage $p_2(\kappa)$, and there must exist some function $\mathrm{negl}''(\kappa)$ such that $p_2(\kappa) < \mathrm{negl}''(\kappa) \implies p_1(\kappa) < \mathrm{negl}''(\kappa) + \mathrm{negl}'(\kappa) \implies p_0(\kappa) < \mathrm{negl}(\kappa)$ where

$$\mathrm{negl}(\kappa) \mapsto \max\left(\frac{|\mathcal{K}| \cdot (\mathrm{negl}''(\kappa) + \mathrm{negl}'(\kappa))}{1 - e^{-1}}, \sqrt{\frac{\mathrm{negl}''(\kappa) + \mathrm{negl}'(\kappa)}{1 - e^{-1}}}\right) \qquad \square$$

Lemma 3.4. *Consider a modified version of the EUF-CMA experiment (Definition 2.3) wherein the adversary is permitted to attempt forgery repeatedly, and freely interleave its attempts with queries to the signing oracle. If at any point the adversary forges successfuly, it immediately wins the game; otherwise the game continues. After attempting to forge on some message m, the adversary can subsequently query the singing oracle on m, but doing so excludes m from future forgery attempts. If Σ is a consistent, EUF-CMA secure signature scheme, then the probability of even one succesful forgery appearing in the sequence of possible forgeries is negligible for any PPT adversary.*

Proof. Consider a further-modified version of the experiment, wherein if a successful forgery (m, σ) is detected in the sequence produced by $\mathcal{G}$, as judged by the output of $b \leftarrow \mathsf{Verify}(m, \sigma)$, then a *second* call $b' \leftarrow \mathsf{Verify}(m, \sigma)$ is performed with fresh randomness, and the experiment outputs b'. The advantage of $\mathcal{G}$ is reduced by a negligible amount relative to the experiment with output b, by reduction to consistency.[21]

$\mathcal{G}$'s success probability in this new experiment can be bounded directly via reduction to EUF-CMA. Let the EUF-CMA adversary $\mathcal{G}'$ run $\mathcal{G}$ and then call Verify to check whether each potential forgery is successful. If a successful forgery is found, then $\mathcal{G}'$ outputs it and exits. Running $\mathcal{G}'$ in the EUF-CMA experiment is identical to running $\mathcal{G}$ in the further-modified experiment. In particular, the second call to Verify in the further-modified experiment corresponds to the challenger's check for a successful forgery in the EUF-CMA experiment. $\square$

Now we are ready to prove that π_Σ UC-realizes $\mathcal{F}_{\mathsf{sig}}$ if Σ achieves a game-based notion of security (i.e., (1) $\implies$ (3) in Theorem 3.2).

Lemma 3.5. *Let Σ be a linear-time signature scheme. If Σ is correct, EUF-CMA secure, and consistent, then for any PPT adversary $\mathcal{A}$ that adaptively corrupts any number of parties, there exists a simulator $\mathcal{S}_{\mathsf{sig}}$ such that*

$$\left\{\textsc{Real}_{\pi_\Sigma, \mathcal{A}, \mathcal{Z}}(\kappa, z)\right\}_{\kappa \in \mathbb{N}, z \in \{0,1\}^*} \approx_{\mathrm{c}} \left\{\textsc{Ideal}_{\mathcal{F}_{\mathsf{sig}}, \mathcal{S}_{\mathsf{sig}}, \mathcal{Z}}(\kappa, z)\right\}_{\kappa \in \mathbb{N}, z \in \{0,1\}^*}.$$

[21] The reduction to consistency must guess the index of the forgery, so it has linear security loss.

Proof. We begin by defining the simulator $\mathcal{S}_{\mathsf{sig}}$:

1. $\mathcal{S}_{\mathsf{sig}}$ emulates $\mathcal{A}$ internally, communicating with the environment on its behalf.
2. Upon receiving $(\mathtt{init}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{sig}}$, $\mathcal{S}_{\mathsf{sig}}$ responds with $(\mathtt{algs}, \mathsf{sid}, \Sigma')$, where $\Sigma' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Verify}')$ is defined by hardcoding the security parameter 1^κ in the three algorithms of Σ and padding its description length such that $\mathcal{F}_{\mathsf{sig}}$ will never terminate its execution and switch to random mode before it produces output.

   More precisely, we pad the size of Σ' to T bits, where T is greater than the number of computational steps required to run $\mathsf{Gen}(1^\kappa)$ and $T \cdot (\ell + 1)$ is greater than the number of computational steps required to run Sign or Verify on security parameter 1^κ and a message of length ℓ.
3. Thereafter, whenever $\mathcal{A}$ indicates that it wishes to corrupt some party P, $\mathcal{S}_{\mathsf{sig}}$ sends $(\mathtt{corrupt}, \mathsf{sid}, P)$ to $\mathcal{F}_{\mathsf{sig}}$, and waits for $(\mathtt{corrupt}, \mathsf{sid}, P, C)$. C contains all of the secret keys and random tapes that $\mathcal{F}_{\mathsf{sig}}$ used to sample the keys and signatures requested by P. Tapes for verification requests performed by P are sampled uniformly. This information is sufficient to explain the view of P to $\mathcal{A}$.

Let q_{G}, q_{S}, and q_{V} be upper bounds on the number of queries made by the environment to the $\mathtt{keygen}$, $\mathtt{sign}$, and $\mathtt{verify}$ interfaces of the protocol (or to $\mathcal{F}_{\mathsf{sig}}$ in the ideal world), respectively. Our argument proceeds via a sequence of hybrid experiments, beginning with the real-world experiment

$$\mathcal{H}_0 = \left\{\textsc{Real}_{\pi_\Sigma, \mathcal{A}, \mathcal{Z}}(\kappa, z)\right\}_{\kappa\in\mathbb{N}, z\in\{0,1\}^*}.$$

We describe each subsequent hybrid experiment in terms of the changes it contains relative to its predecessor, and then present an argument that the two remain indistinguishable.

Hybrid $\mathcal{H}_1$. In this experiment, the variables $\mathsf{rmode} := 0, \ell_{\mathsf{pk}} := 1, \ell_{\mathsf{sig}} := 1, \mathcal{K} := \emptyset$, and $\mathcal{Q} := \emptyset$ are defined from the start. rmode is fixed for the duration of the experiment, but the other variables are updated exactly as specified in $\mathcal{F}_{\mathsf{sig}}$. $\mathcal{H}_1$ also introduces all the code of $\mathcal{F}_{\mathsf{sig}}$ for the case that $\mathsf{rmode} = 1$, but this code is never activated.[22] It is therefore the case that $\mathcal{H}_1 = \mathcal{H}_0$.

Hybrid $\mathcal{H}_2$. In this experiment, whenever $\mathcal{Z}$ sends $(\mathtt{keygen}, \mathsf{sid})$ as input to one of the parties P, the experiment saves the generated key $(\mathsf{sk}, \mathsf{pk})$ and the random tape r_k used to sample it in a record $(\mathtt{key}, \mathsf{sid}, P, \mathsf{pk}, \mathsf{sk}, r_k)$. Whenever $\mathcal{Z}$ sends $(\mathtt{sign}, \mathsf{sid}, \mathsf{pk}, m)$ as input to one of the parties P, and P produces a signature as a result,[23] the experiment saves the generated signature σ and the random tape r_σ used to sample it in a record $(\mathtt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, r_\sigma)$. Since none of this behavior is observable by $\mathcal{Z}$, we have $\mathcal{H}_2 = \mathcal{H}_1$.

Hybrid $\mathcal{H}_3$. In this experiment, we implement the public-key uniqueness constraint from Step 4a of $\mathcal{F}_{\mathsf{sig}}$. whenever $\mathcal{Z}$ sends $(\mathtt{keygen}, \mathsf{sid})$ as input to one of the parties, the experiment checks whether $\mathsf{pk} \in \mathcal{K}$, and sets $\mathsf{rmode} := 1$ if so. Note that when a party is corrupted, $\mathcal{A}$ and $\mathcal{Z}$ are able to act on behalf of the corrupted parties without actually invoking them. This check is *only* performed when a party (be it honest or corrupt) is actually invoked.[24] The probability that $\mathsf{rmode} = 1$ at

[22] We note at this point that the contents of this code are not important, since we will prove that it is not activated with more than negligible probability in any of our hybrid experiments. For the sake of simplicity, we can assume that the environment *always* distinguishes successfully when $\mathsf{rmode} = 1$.

[23] P might not produce a signature if the request is invalid.

[24] This also applies to other checks added during the remainder of this proof; hereafter we leave it implicit.

any point is at most negigible, or else there exists a reduction that contradicts Lemma 3.3: the reduction simply emulates $\mathcal{H}_3$ until $\mathsf{rmode} = 1$ and then outputs $\mathcal{K}$. Thus we have $\mathcal{H}_3 \approx_{\mathrm{c}} \mathcal{H}_2$.

Hybrid $\mathcal{H}_4$. In this experiment we implement the first constraint from Step 5a of $\mathcal{F}_{\mathsf{sig}}$: two messages cannot have the same signature under the same public key. Whenever $\mathcal{Z}$ sends $(\mathtt{sign}, \mathsf{sid}, \mathsf{pk}, m)$ as input to one of the parties P, the experiment iterates over all records $(\mathtt{sig}, \mathsf{sid}, \mathsf{pk}, m', \sigma', *)$ such that $m' \neq m$, and if $\mathsf{Verify}(1^\kappa, \mathsf{pk}, m, \sigma') = 1$ for any of them, then the experiment sets $\mathsf{rmode} \coloneqq 1$. If $\mathsf{Verify}(1^\kappa, \mathsf{pk}, m, \sigma') = 0$ for all of them, then P proceeds to generate a signature on m as it did in $\mathcal{H}_3$. The environment can only distinguish if $\mathsf{Verify}(1^\kappa, \mathsf{pk}, m, \sigma') = 1$ for some σ' previously issued on $m' \neq m$, so we bound the probability of this event occurring by reduction to Lemma 3.4.

The reduction emulates $\mathcal{H}_4$, samples $i \leftarrow [q_{\mathsf{G}}]$, and on the i^{th} event that $\mathcal{Z}$ sends $(\mathtt{keygen}, \mathsf{sid})$ to some party (let this party be P), the experiment programs that party's output to be the challenge instance pk. Subsequently, when $\mathcal{Z}$ sends $(\mathtt{sign}, \mathsf{sid}, \mathsf{pk}, m)$ to P, if it is the *first* time a request has been issued on m under pk, then for every signature σ' that the reduction previously received from its signing oracle, the reduction adds (m, σ') to its stream of potential forgeries. The reduction then invokes the signing oracle q_{S} times to build a cache of q_{S} signatures on m; these are used to answer this signing request and all subsequent signing requests on m under pk.

Recall that in the forgery game (both the original one from Definition 2.3 and the "stream-based" version presented in Lemma 3.4), a forgery on m is only counted as successful if the signing oracle has never previously been queried on m. On the other hand, the environment distinguishes $\mathcal{H}_4$ from $\mathcal{H}_3$ if a signature verifies successfully on two different messages m and m', regardless of the order in which they are signed or whether they have been signed previously. The reduction's use of cached signatures induces a total ordering on messages and guarantees that the environment's distinguishing condition implies its own forgery condition[25] (assuming it has guessed the correct public key pk). Therefore, the reduction has advantage in contradicting Lemma 3.4 that is no less than $\varepsilon/q_{\mathsf{G}}$, where ε is the environment's advantage in distinguishing. It follows that $\mathcal{H}_4 \approx_{\mathrm{c}} \mathcal{H}_3$.

Hybrid $\mathcal{H}_5$. This hybrid is like $\mathcal{H}_4$, except that when a $(\mathtt{sign}, \mathsf{sid}, \mathsf{pk}, m)$ instruction is issued and the experiment finds some record $(\mathtt{sig}, \mathsf{sid}, \mathsf{pk}, m', \sigma', *)$ such that $m' \neq m$ and $\mathsf{Verify}(1^\kappa, \mathsf{pk}, m, \sigma') = 1$, it then waits for the signing party P to run $\sigma \leftarrow \mathsf{Sign}(1^\kappa, \mathsf{sk}, m)$ and sets $\mathsf{rmode} \coloneqq 1$ *only if* $\sigma = \sigma'$ (and otherwise continues as though the record was not found). The conditions that cause $\mathsf{rmode} = 1$ in $\mathcal{H}_5$ are a subset of the conditions that cause $\mathsf{rmode} = 1$ in $\mathcal{H}_4$ and therefore $\mathcal{H}_4 \approx_{\mathrm{c}} \mathcal{H}_3 \implies \mathcal{H}_5 \approx_{\mathrm{c}} \mathcal{H}_3$.

Hybrid $\mathcal{H}_6$. In this experiment, whenever $\mathcal{Z}$ sends $(\mathtt{sign}, \mathsf{sid}, \mathsf{pk}, m)$ as input to one of the parties P, P *first* runs $\sigma \leftarrow \mathsf{Sign}(1^\kappa, \mathsf{sk}, m)$, and *then* the experiment checks whether $\mathsf{Verify}(1^\kappa, \mathsf{pk}, m, \sigma) = 0$ or whether there is some record $(\mathtt{sig}, \mathsf{sid}, \mathsf{pk}, m', \sigma', *)$ such that $m \neq m'$ and $\sigma = \sigma'$ and sets $\mathsf{rmode} \coloneqq 1$ if either of these conditions hold. $\mathcal{H}_6$ differs from $\mathcal{H}_5$ only when $\mathsf{Verify}(1^\kappa, \mathsf{pk}, m, \sigma) = 0$, and by a simple reduction to the correctness of Σ (Definition 2.2), this happens with at most negligible probability. Therefore $\mathcal{H}_6 \approx_{\mathrm{c}} \mathcal{H}_5$.

Hybrid $\mathcal{H}_7$. This experiment differs from $\mathcal{H}_6$ only in the case that a party P is corrupted. In $\mathcal{H}_6$, the internal state of P is communicated directly to the adversary. This state comprises the random tapes that P used when it executed the Gen, Sign, and Verify algorithms.[26] In $\mathcal{H}_7$, when

[25] Specifically, all queries to the signing oracle that will ever be made for some message m' are made at the same time, and their results are available immediately for use in judging forgeries on m *before* any signing requests are made on m itself.

[26] Note that the extra calls to Verify that have been added during $\mathtt{sign}$ are performed by the *experiment*, not by

$\mathcal{A}$ requests to corrupt P, it is given the tapes from the relevant `key` and `sig` records that are kept by the experiment (these tapes are precisely the ones used by P), and for every past invocation of $\mathsf{Verify}(1^\kappa, \mathsf{pk}, m, \sigma; r)$ by P, the experiment samples a *new* tape r' at the time of corruption and transmits it to $\mathcal{A}$, instead of revealing the tape r that P actually used. This corresponds to implementing Step 7 of $\mathcal{F}_{\mathsf{sig}}$. The only value that depends upon r and is known to $\mathcal{A}$ *before* it corrupts P is the output of $\mathsf{Verify}(1^\kappa, \mathsf{pk}, m, \sigma; r)$. Therefore, $\mathcal{H}_7$ can be distinguished from $\mathcal{H}_6$ only if $\mathsf{Verify}(1^\kappa, \mathsf{pk}, m, \sigma; r') \neq \mathsf{Verify}(1^\kappa, \mathsf{pk}, m, \sigma; r)$. We will show by reduction to the consistency of Σ (Definition 2.4) that this happens with negligible probability.

Let $(\texttt{verify}, \mathsf{sid}, \mathsf{pk}_i, m_i, \sigma_i)$ for $i \in [q_{\mathsf{V}}]$ be the i^{th} `verify` instruction issued by $\mathcal{Z}$ to any party in $\mathcal{H}_7$. The reduction emulates $\mathcal{H}_7$ internally, and afterward samples $i \leftarrow [q_{\mathsf{V}}]$ and outputs $(\mathsf{pk}_i, m_i, \sigma_i)$ to the challenger. The probability ε that $\mathcal{Z}$ distinguishes $\mathcal{H}_7$ from $\mathcal{H}_6$ successfully is upper bounded by the probability that there is *at least* one $i \in [q_{\mathsf{V}}]$ such that $\mathsf{Verify}(1^\kappa, \mathsf{pk}_i, m_i, \sigma_i; r'_i) \neq \mathsf{Verify}(1^\kappa, \mathsf{pk}_i, m_i, \sigma_i; r_i)$. Thus if $\varepsilon_i = \Pr[\mathsf{Verify}(1^\kappa, \mathsf{pk}_i, m_i, \sigma_i; r'_i) \neq \mathsf{Verify}(1^\kappa, \mathsf{pk}_i, m_i, \sigma_i; r_i)]$, then we have $\varepsilon \leq \sum_{i \in [q_{\mathsf{V}}]} \varepsilon_i$ which implies the expected success probability for the reduction over a random choice of i is no less than $\varepsilon / q_{\mathsf{V}}$. If Σ is consistent and q_{V} is at most polynomial in κ, then ε must be negligible.

Hybrid $\mathcal{H}_8$. In this experiment we effectively implement the correctness constraint from Step 6a of $\mathcal{F}_{\mathsf{sig}}$. Whenever $\mathcal{Z}$ sends $(\texttt{verify}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ as input to one of the parties P, if a record of the form $(\texttt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, *)$ exists in memory, then P skips running Verify and behaves as though its output was 1. Such records *only* exist if σ was produced by $\mathsf{Sign}(1^\kappa, \mathsf{sk}, m; r_\sigma)$ for some uniformly sampled r_σ, and therefore $\mathcal{H}_8 \approx_{\mathrm{c}} \mathcal{H}_7$ by a simple reduction to the correctness of Σ (Definition 2.2).

Hybrid $\mathcal{H}_9$. In this experiment we effectively implement the consistency constraint from Steps 6a and 6c of $\mathcal{F}_{\mathsf{sig}}$. Whenever $\mathcal{Z}$ sends $(\texttt{verify}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ as input to one of the parties P, the experiment records the result in the form of a `sig` or `bad-sig` record. Specifically, if P returns $(\texttt{verified}, \mathsf{sid}, \mathsf{pk}, m, \sigma, 1)$ to $\mathcal{Z}$ and no record of the form $(\texttt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, *)$ exists, then the experiment records $(\texttt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, \bot)$. If P returns $(\texttt{verified}, \mathsf{sid}, \mathsf{pk}, m, \sigma, 0)$ to $\mathcal{Z}$ and $(\texttt{bad-sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ is not recorded, then the experiment records it. Additionally, when $\mathcal{Z}$ sends $(\texttt{verify}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ as input to one of the parties P, if $(\texttt{bad-sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ is recorded, then P skips running Verify and behaves as though its output was 0.

$\mathcal{H}_9$ is distinguished from $\mathcal{H}_8$ only by the case that a $(\texttt{verify}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ instruction is given *twice* by $\mathcal{Z}$ on a σthat was *not* produced by a `sign` instruction. In $\mathcal{H}_9$, the probability of two different results is exactly 0, whereas in $\mathcal{H}_8$, it is nonzero. Using a variation of the reduction and argument that we introduced in the context of $\mathcal{H}_7$, we can show that if Σ is consistent and q_{V} is at most polynomial in κ, then $\mathcal{H}_8 \approx_{\mathrm{c}} \mathcal{H}_9$.

Hybrid $\mathcal{H}_{10}$. Recall that when a $(\texttt{sign}, \mathsf{sid}, \mathsf{pk}, m)$ instruction is issued in all hybrids since $\mathcal{H}_6$, after party P computes $\sigma \leftarrow \mathsf{Sign}(1^\kappa, \mathsf{sk}, m)$, the experiment immediately checks whether $\mathsf{Verify}(1^\kappa, \mathsf{pk}, m, \sigma) = 0$ an sets $\mathsf{rmode} := 1$ if so. In $\mathcal{H}_{10}$, after party P invokes $\sigma \leftarrow \mathsf{Sign}(1^\kappa, \mathsf{sk}, m)$, the experiment first checks whether the record $(\texttt{bad-sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ exists in memory, and if it does, then the experiment behaves as though Verify had returned 0, without actually calling Verify.

In $\mathcal{H}_{10}$, the record $(\texttt{bad-sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ is only stored if the experiment has confirmed that $\mathsf{Verify}(1^\kappa, \mathsf{sid}, \mathsf{pk}, m, \sigma) = 0$ during the processing of a $(\texttt{verify}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ instruction. $\mathcal{H}_{10}$ and $\mathcal{H}_9$ can therefore be distinguished only by a $\mathcal{Z}$ that predicts a σ that *will* be issued by Sign on some m under some pk and then instructs some party to verify (pk, m, σ) before the issuance actually happens. In $\mathcal{H}_{10}$, such a sequence will always result in $\mathsf{rmode} = 1$, whereas in $\mathcal{H}_9$ there is a nonzero

P, and so these tapes are excluded.

chance that $\mathsf{Verify}(1^\kappa, \mathsf{pk}, m, \sigma) = 1$ during the $\mathtt{sign}$ instruction, and rmode remains 0. Once again, we can use a variation of the reduction and argument that we introduced in the context of $\mathcal{H}_7$ to show that if Σ is consistent and q_V is at most polynomial in κ, then $\mathcal{H}_9 \approx_\mathrm{c} \mathcal{H}_{10}$.

Hybrid $\mathcal{H}_{11}$. In this experiment, whenever $\mathcal{Z}$ sends $(\mathtt{verify}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ as input to one of the parties P, if $(\mathtt{key}, \mathsf{sid}, P', \mathsf{pk}, \mathsf{sk}, r_k)$ is recorded, P' is not corrupted, and the experiment has not stored any records of the form $(\mathtt{sig}, \mathsf{sid}, \mathsf{pk}, m, *, *)$,[27] then the experiment skips running Verify and behaves as though its output was 0. In particular, it stores the relevant $\mathtt{bad\text{-}sig}$ record, preventing σ from being issued on m in the future; this essentially implements the second constraint from Step 6b of $\mathcal{F}_\mathsf{sig}$.

$\mathcal{H}_{11} \approx_\mathrm{c} \mathcal{H}_{10}$ by reduction to Lemma 3.4. The reduction emulates $\mathcal{H}_{11}$, samples $i \leftarrow [q_\mathsf{G}]$, and on the i^{th} event that $\mathcal{Z}$ sends $(\mathtt{keygen}, \mathsf{sid})$ to some party (let this party be P), the experiment programs that party's output to be the challenge instance pk. Subsequently, when $\mathcal{Z}$ sends $(\mathtt{sign}, \mathsf{sid}, \mathsf{pk}, m)$ to P, the reduction uses its signing oracle to produce responses. If P is ever corrupted, then the reduction aborts. Whenever $\mathcal{Z}$ sends $(\mathtt{verify}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ to some party P' such that m was never signed under pk, the reduction adds (m, σ) to its stream of potential forgeries. The environment's only means of distinguishing $\mathcal{H}_{11}$ from $\mathcal{H}_{10}$ is to produce an actual forgery under some honest party's public key, and request verification for it; conditioned on the reduction guessing the correct public key, any such actual forgery is always included in the reduction's stream of potential forgeries. Therefore, the reduction has advantage in contradicting Lemma 3.4 that is no less than $\varepsilon / q_\mathsf{G}$, where ε is the environment's advantage in distinguishing $\mathcal{H}_{11}$ from $\mathcal{H}_{10}$.

Hybrid $\mathcal{H}_{12}$. This experiment is like $\mathcal{H}_{11}$, except that after party P computes $\sigma \leftarrow \mathsf{Sign}(1^\kappa, \mathsf{sk}, m)$, the experiment checks whether the record $(\mathtt{bad\text{-}sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ exists, and if it does not, then instead of running Verify as in $\mathcal{H}_{11}$, the experiment behaves as though Verify had returned 1. In $\mathcal{H}_{12}$, Verify is not used at all during the processing of $\mathtt{sign}$ instructions. $\mathcal{H}_{12} \approx_\mathrm{c} \mathcal{H}_{11}$ by a simple reduction to the correctness of Σ (Definition 2.2).

Hybrid $\mathcal{H}_{13}$. This experiment defines $\Sigma' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Verify}')$ by hardcoding the security parameter 1^κ into the three algorithms of Σ and padding the description of Σ' with no-op instructions until it is T bits, where T is greater than the number of computational steps required to run $\mathsf{Gen}(1^\kappa)$ and $T \cdot (\ell + 1)$ is greater than the number of computational steps required to run Sign or Verify on security parameter 1^κ and a message of length ℓ. Due to the fact that Σ is PPT in κ and linear time in its other inputs (Definition 2.5), such a T is guaranteed to exist. Throughout $\mathcal{H}_{13}$, Gen' is used in place of Gen, Sign' in place of Sign, and Verify' in place of Verify. Since the behavior of the new functions is exactly the same as those that they replaced, we have $\mathcal{H}_{13} = \mathcal{H}_{12}$.

Hybrid $\mathcal{H}_{14}$. This experiment finally bridges the gap to the ideal world. The code of "the experiment" in $\mathcal{H}_{13}$ is completely removed, and the code of the parties is removed and replaced with dummy-party code that invokes $\mathcal{F}_\mathsf{sig}$. The simulator $\mathcal{S}_\mathsf{sig}$ (defined at the beginning of this proof) replaces $\mathcal{A}$, supplies Σ' on request, and handles the sampling of verification tapes when a corruption occurs. The behavior of $\mathcal{H}_{14}$ is completely defined by $\mathcal{F}_\mathsf{sig}$, $\mathcal{S}_\mathsf{sig}$, and the dummy parties; in other words,

$$\mathcal{H}_{14} = \left\{ \textsc{Ideal}_{\mathcal{F}_\mathsf{sig}, \mathcal{S}_\mathsf{sig}, \mathcal{Z}}(\kappa, z) \right\}_{\kappa \in \mathbb{N}, z \in \{0,1\}^*}.$$

It remains only to observe that due to the padding of Σ', $\mathcal{F}_\mathsf{sig}$ will never set $\mathsf{rmode} = 1$ due to a failure of Gen', Sign', or Verify' to terminate within the time bound. Therefore, we also have

[27] Such a record would indicate that that a $(\mathtt{sign}, \mathsf{sid}, \mathsf{pk}, m)$ command was previously issued to P' by $\mathcal{Z}$.

$\mathcal{H}_{14} = \mathcal{H}_{13}$, and Lemma 3.5 follows by transitivity. $\square$

Next we prove that π_Σ *only* UC-realizes $\mathcal{F}_{\mathsf{sig}}$ if Σ achieves a game-based notion of security (i.e., (2) $\implies$ (1) in Theorem 3.2).

Lemma 3.6. *Let Σ be a linear-time signature scheme. If Σ is* not *correct, EUF-CMA secure, and consistent, then there exists a PPT adversary $\mathcal{A}$ that corrupts no parties, such that for any PPT simulator $\mathcal{S}$ there exists a PPT environment $\mathcal{Z}$ that has a non-negligible advantage in distinguishing*

$$\left\{\text{REAL}_{\pi_\Sigma,\mathcal{A},\mathcal{Z}}(\kappa, z)\right\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*} \qquad \textit{from} \qquad \left\{\text{IDEAL}_{\mathcal{F}_{\mathsf{sig}},\mathcal{S}_{\mathsf{sig}},\mathcal{Z}}(\kappa, z)\right\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}.$$

Proof. For each desired property of Σ, and for any adversary $\mathcal{G}$ attacking this property of Σ, we show how to construct an environment $\mathcal{Z}$ and adversary $\mathcal{A}$ such that for any ideal adversary $\mathcal{S}$, the environment $\mathcal{Z}$ can distinguish between a real execution of π_Σ with $\mathcal{A}$ and an ideal execution of $\mathcal{F}_{\mathsf{sig}}$ with $\mathcal{S}$, with advantage equal to the advantage of $\mathcal{G}$.

Correctness. Let $\mathcal{G}$ be an adversary attacking the correctness of Σ. The environment $\mathcal{Z}$ activates an uncorrupted party P with input $(\texttt{keygen}, \mathsf{sid})$, and receive $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk})$ in response. $\mathcal{Z}$ then runs $\mathcal{G}(1^\kappa, \mathsf{pk})$, and for each m generated by $\mathcal{G}$, it activates the P again with input $(\texttt{sign}, \mathsf{sid}, \mathsf{pk}, m)$ and receives a signature $(\texttt{signature}, \mathsf{sid}, \mathsf{pk}, \sigma)$. $\mathcal{Z}$ then activates P with $(\texttt{verify}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ and obtains a bit b indicating whether σ has verified successfully. If $b = 0$, $\mathcal{Z}$ outputs 1, and otherwise it passes σ to $\mathcal{G}$ and continues running. When $\mathcal{G}$ terminates, $\mathcal{Z}$ outputs 0.

In the ideal world $\mathcal{Z}$ will always output 0, because $\mathcal{F}_{\mathsf{sig}}$ will record $(\texttt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, r_\sigma)$ when processing $\texttt{sign}$, and later check for this record while processing $\texttt{verify}$ so that it outputs $b = 1$. In the real world, the view of $\mathcal{G}$ is exactly the same as its view in the correctness experiment (see Definition 2.2), and the output of $\mathcal{Z}$ is distributed identically to the experiment's output.

Existential Unforgeability. Let $\mathcal{G}$ be an adversary attacking the EUF-CMA security of Σ. $\mathcal{Z}$ activates an uncorrupted party P with input $(\texttt{keygen}, \mathsf{sid})$, receives $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk})$ in response, and gives pk to $\mathcal{G}$ as the challenge public key. Upon receiving a signing query for m' from $\mathcal{G}$, $\mathcal{Z}$ activates P with input $(\texttt{sign}, \mathsf{sid}, \mathsf{pk}, m')$, receives $(\texttt{signature}, \mathsf{sid}, \mathsf{pk}, \sigma')$, and sends σ' in response to $\mathcal{G}$. When $\mathcal{G}$ outputs its forgery (m^*, σ^*), $\mathcal{Z}$ checks if m^* has been queried by $\mathcal{G}$. If it has, then, then $\mathcal{Z}$ outputs 0. Otherwise, $\mathcal{Z}$ activates some party P' with input $(\texttt{verify}, \mathsf{sid}, \mathsf{pk}, m^*, \sigma^*)$, receives a bit b indicating whether verification was successful, and outputs b. In an ideal execution with $\mathcal{F}_{\mathsf{sig}}$, if a particular message has never been signed under an honest party's public key, then no signature will ever verify under that public key, and consequently $\mathcal{Z}$ will always output 0. In a real execution, $\mathcal{G}$'s view is exactly the same as in the EUF-CMA security game (see Definition 2.3), thus if $\mathcal{G}$ produces a forgery (m^*, σ^*) that successfully verifies (winning the game) then $\mathcal{Z}$ outputs 1.

Consistency. Let $\mathcal{G}$ be an adversary that generates some (pk, m, σ) with the intent of causing Verify to produce inconsistent outputs. $\mathcal{Z}$ runs $(\mathsf{pk}, m, \sigma) \leftarrow \mathcal{G}(1^\kappa)$, and then activates some honest party twice with input $(\texttt{verify}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$. Let b and b' the results of these two verifications. $\mathcal{Z}$ outputs $b \oplus b'$. In the ideal world, $\mathcal{Z}$ will always output 0, because $\mathcal{F}_{\mathsf{sig}}$ caches the results of verification queries. In the real world, $\mathcal{Z}$'s verification test is exactly the same as the challenger's test in the consistency game (see Definition 2.4), and thus $\mathcal{Z}$'s output is distributed identically to the game's output. $\square$

3.3 Extracting a Signature Scheme from Any UC-Secure Signature Protocol

So far we have shown in Section 3.2 that for protocols structured as π_{Σ} for some signature scheme Σ, UC-realizing $\mathcal{F}_{\mathsf{sig}}$ is equivalent to satisfying the property-based signature definition that we gave in Section 2.1. However, there might be other realizations of $\mathcal{F}_{\mathsf{sig}}$ that do not fit into the framework of a signature scheme. We now show that from any such realization in the plain model, we can extract a Σ such that π_{Σ} also UC-realizes $\mathcal{F}_{\mathsf{sig}}$ (with security against static corruptions). Combined with Theorem 3.2, this theorem implies that plain model instantiations of $\mathcal{F}_{\mathsf{sig}}$ always correspond to signature schemes.

Theorem 3.7. *For any protocol π_{sig} that UC-realizes $\mathcal{F}_{\mathsf{sig}}$ in the plain model with security against any number of static malicious corruptions, there exists a linear-time signature scheme $\Sigma_{\mathcal{S}}$ such that the real-world experiments involving $\pi_{\Sigma_{\mathcal{S}}}$ and π_{sig} are indistinguishable in the presence of any number of static malicious corruptions.*

Before proving this theorem, we present a useful corollary as motivation.

Corollary 3.8. *For any protocol π_{sig} that UC-realizes $\mathcal{F}_{\mathsf{sig}}$ in the plain model with security against any number of static malicious corruptions, if $\Sigma_{\mathcal{S}}$ is the linear-time signature scheme from Theorem 3.7, then $\Sigma_{\mathcal{S}}$ is correct, EUF-CMA secure, and consistent, and $\pi_{\Sigma_{\mathcal{S}}}$ UC-realizes $\mathcal{F}_{\mathsf{sig}}$ in the plain model with security against any number of* adaptive *malicious corruptions.*

Proof of Corollary 3.8. Apply Theorem 3.7, then use the fact that π_{sig} UC-realizes $\mathcal{F}_{\mathsf{sig}}$ to change from the real world to the ideal world, and finally apply Theorem 3.2. $\square$

Next, we prove a lemma asserting random mode can be distinguished from non-random mode, regardless of the behavior of Σ. Afterward, we present our proof of Theorem 3.7, which makes use of this lemma.

Lemma 3.9. *For any protocol π_{sig} and any environment and ideal adversary $(\mathcal{Z}, \mathcal{S})$ that cause $\mathcal{F}_{\mathsf{sig}}$ to enter random mode with probability p, there exists an environment $\mathcal{Z}'$ that distinguishes the real experiment involving π_{sig} from the ideal one involving $\mathcal{F}_{\mathsf{sig}}$ with advantage at least $p/2$.*

Proof. Let $\mathcal{Z}'$ emulate the original environment $\mathcal{Z}$ internally and forwards its instructions to and from the other entities in the experiment. In addition to the parties invoked by $\mathcal{Z}$, $\mathcal{Z}'$ creates two honest parties, P_{S} and P_{V}. $\mathcal{Z}'$ will arrange the experiment such that neither $\mathcal{Z}$ nor the adversary (real or ideal) ever becomes aware of these additional parties. We observe first that there is no circumstance under which $\mathcal{F}_{\mathsf{sig}}$ communicates with $\mathcal{S}$ after the first $(\mathtt{init}, \mathsf{sid})$ instruction is issued for a particular sid. We observe second that the $\mathtt{keygen}$, $\mathtt{sign}$, and $\mathtt{verify}$ interfaces of $\mathcal{F}_{\mathsf{sig}}$ do not permit the adversary to *delay* output to the party that invoked them. This means that if π_{sig} realizes $\mathcal{F}_{\mathsf{sig}}$ in the plain UC model (which is asynchronous), then π_{sig} never waits on inbound communication during the processing of those instructions.[28] These facts together imply that after the first $\mathtt{init}$ instruction is issued, the environment can activate a party in π_{sig} and receive output from it without ever activating any other entity in the experiment, including the adversary.[29]

$\mathcal{Z}'$ watches the commands issued by $\mathcal{Z}$ and waits until it issues $(\mathtt{init}, \mathsf{sid})$ to some party (that is neither P_{S} nor P_{V}). $\mathcal{Z}'$ continues to observe the instructions issued by $\mathcal{Z}$ in the session associated with sid, and records the messages signed in the set $\mathcal{M}$. When $\mathcal{Z}$ halts, $\mathcal{Z}'$ samples a random

[28] If it did, then the adversary could cause a delay.

[29] Since it is legal for parties in π_{sig} to *send* messages, and they might return activation to the environment when they do so, the environment may have to activate a party several times in order to receive output. Nevertheless, no other entity needs to be activated.

message $m \leftarrow \{0,1\}^\kappa \setminus \mathcal{M}$ that has never been signed under sid,[30] and flips a coin $c \leftarrow \{0,1\}$ to choose between two different sampling methods for (pk, σ):

- If $c = 0$, then $\mathcal{Z}'$ activates P_S with input $(\mathtt{keygen}, \mathsf{sid})$ and receives $(\mathtt{public\text{-}key}, \mathsf{sid}, \mathsf{pk})$ in response, after which it activates P_S with input $(\mathtt{sign}, \mathsf{sid}, \mathsf{pk}, m)$ and receives $(\mathtt{signature}, \mathsf{sid}, \mathsf{pk}, \sigma)$ in response.
- If $c = 1$, then $\mathcal{Z}'$ internally emulates the real world π_sig running $(\mathtt{keygen}, \mathsf{sid})$ and $(\mathtt{sign}, \mathsf{sid}, \mathsf{pk}, m)$ as P_S. Note that it does so without actually invoking P_S, and this is possible because P_S has never been invoked (nor has it received any messages) and therefore cannot possibly have secret state.

Finally, $\mathcal{Z}'$ activates P_V with input $(\mathtt{verify}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ receives $(\mathtt{verified}, \mathsf{sid}, \mathsf{pk}, m, \sigma, b)$ in response, and outputs $c \oplus b$.

In the real world, P_S runs π_sig, and the view of P_V is independent of c because pk and σ are generated in the same way, either by P_S or by $\mathcal{Z}'$ emulating P_S internally. Therefore, $\mathcal{Z}'$ outputs 1 with probability $1/2$.

In the ideal world, if $c = 0$ then the verification always returns $b = 1$, because the $\mathtt{sign}$ message causes $\mathcal{F}_\mathsf{sig}$ to store a record $(\mathtt{sig}, \mathsf{sid}, \mathsf{pk}, m, \sigma, r_\sigma)$, which it checks when processing $\mathtt{verify}$. If $c = 1$ in the ideal world, and $\mathcal{F}_\mathsf{sig}$ is in random mode, then verification always returns $b = 0$, since $\mathcal{Z}'$ samples m explicitly from the set of previously-unsigned messages and never communicates it to $\mathcal{F}_\mathsf{sig}$, and $\mathcal{F}_\mathsf{sig}$ always returns $b = 0$ for messages it has not seen, in random mode. If $c = 1$ in the ideal world, and $\mathcal{F}_\mathsf{sig}$ is *not* in random mode, then verification returns $b = 1$ with probability $1 - \mathrm{negl}(\kappa)$. In this case, $\mathcal{Z}'$ generated pk by emulating π_sig internally and never communicated it to $\mathcal{F}_\mathsf{sig}$, which means that $\mathcal{F}_\mathsf{sig}$ has stored a $\mathtt{key}$ record containing pk with at most negligible probability (this can be established via a simple reduction to Lemma 3.3). When $\mathcal{F}_\mathsf{sig}$ receives a verification request on an unknown public key pk, it uses the Verify algorithm to determine the value of b, and thus by the correctness of Σ (see Definition 2.2), we have $b = 1$ with all but negligible probability.

Therefore, in the ideal world, $\mathcal{Z}'$ outputs 1 with probability $1/2 + p/2 + \mathrm{negl}(\kappa)$. It follows that the advantage of $\mathcal{Z}'$ is $p/2 + \mathrm{negl}(\kappa)$. □

Proof of Theorem 3.7. By the assumption that π_sig UC-realizes $\mathcal{F}_\mathsf{sig}$, there must exist a simulator $\mathcal{S}$ that simulates the dummy adversary (which simply forwards messages between $\mathcal{Z}$ and entities in the real protocol) to $\mathcal{Z}$. By Lemma 3.9, we can be sure that $\mathcal{S}$ causes $\mathcal{F}_\mathsf{sig}$ to enter random mode with at most negligible probability. Next, we describe how to extract the signature scheme $\Sigma_\mathcal{S}$ from the simulator $\mathcal{S}$:

Algorithm 3.10. $\Sigma_\mathcal{S}$ (A Signature Scheme from the Simulator $\mathcal{S}$ for π_sig and $\mathcal{F}_\mathsf{sig}$)

Let sid be some fixed valid session identifier for $\mathcal{S}$. If $\mathcal{F}_\mathsf{sig}$ sends $(\mathtt{init}, \mathsf{sid})$ to $\mathcal{S}$, then $\mathcal{S}$ must immediately respond with some algorithms $(\mathtt{algs}, \mathsf{sid}, \Sigma')$ with overwhelming probability. Let $\mathcal{S}_\mathtt{algs}(1^\kappa; r_\mathcal{S})$ be an algorithm that outputs $(\mathsf{Gen}', \mathsf{Sign}', \mathsf{Verify}') = \Sigma'$ by emulating $\mathcal{F}_\mathsf{sig}$ in order to run $\mathcal{S}$ in this way, with random tape $r_\mathcal{S}$.[a] Let $\ell(\kappa)$ be an upper bound on the bit-length of the random tape requred by $\mathcal{S}_\mathtt{algs}$, which must exist because $\mathcal{S}$ is PPT. Let $p(\kappa)$ be an upper bound on both the size of the description of Σ' and the sizes of the keys pk and sk' produced by Gen', which must exist because $\mathcal{S}$ and Gen' are both PPT. Let $T(\kappa) \mapsto p(\kappa) \cdot (p(\kappa) + 1)$

[30]Since $\mathcal{Z}$ makes at most polynomially many signing requests, there must exist such a message.

Finally, let $\mathsf{Limit}_x(F)$ modify a function F to run as normal for x steps, and then output either F's output, or 0 if F has not finished.

Gen(1^κ) :

```
1. $r_{\mathcal{S}} \leftarrow \{0,1\}^{\ell(\kappa)}$
2. $(\mathsf{Gen}', \mathsf{Sign}', \mathsf{Verify}') := \mathcal{S}_{\texttt{algs}}(1^\kappa; r_{\mathcal{S}})$
3. $(\mathsf{sk}', \mathsf{pk}) \leftarrow \mathsf{Gen}'()$
4. $\mathsf{sk} := (\mathsf{sk}', r_{\mathcal{S}})$
5. output $(\mathsf{sk}, \mathsf{pk})$
```

Sign$(1^\kappa, \mathsf{sk}, m)$:

```
6. $(\mathsf{sk}', r_{\mathcal{S}}) := \mathsf{sk}$
7. $(\mathsf{Gen}', \mathsf{Sign}', \mathsf{Verify}') := \mathcal{S}_{\texttt{algs}}(1^\kappa; r_{\mathcal{S}})$
8. $\sigma \leftarrow \mathsf{Limit}_{T(\kappa)\cdot(|m|+1)}\mathsf{Sign}'(\mathsf{sk}', m)$
9. output $\sigma$
```

Verify$(1^\kappa, \mathsf{pk}, m, \sigma)$:

```
10. $(\mathsf{Gen}', \mathsf{Sign}', \mathsf{Verify}') \leftarrow \mathcal{S}_{\texttt{algs}}(1^\kappa)$
11. $b \leftarrow \mathsf{Limit}_{T(\kappa)\cdot(|m|+|\sigma|+1)}\mathsf{Verify}'(\mathsf{pk}, m, \sigma)$
12. output $b$
```

[a] In this emulated experiment, $\mathcal{S}$ sees no communication from any entity other than $\mathcal{F}_{\mathsf{sig}}$. This does not cause a problem because the quantifier order of UC insists that $\mathcal{S}$ must work for every PPT $\mathcal{Z}$, including an $\mathcal{Z}$ that does not activate $\mathcal{S}$ before $\mathcal{F}_{\mathsf{sig}}$ does.

The main difficulty with building $\Sigma_{\mathcal{S}}$ from $\mathcal{S}_{\texttt{algs}}(1^\kappa; r_{\mathcal{S}})$ is the random tape $r_{\mathcal{S}}$, which might influence the descriptions of Σ'. We have no means to make all parties in the system agree on $r_{\mathcal{S}}$, and even if they *did* agree, this would imply publishing Σ', which might cause problems if the description of Σ' contains sensitive information that allows the environment to distinguish π_{sig} from the ideal experiment involving $\mathcal{F}_{\mathsf{sig}}$.[31] Instead, we sample $r_{\mathcal{S}}$ afresh for each signing key, and keep it as part of the secret key $\mathsf{sk} := (\mathsf{sk}', r_{\mathcal{S}})$. Later, during signing, we resuse $r_{\mathcal{S}}$ to get the same set of algorithms from $\mathcal{S}_{\texttt{algs}}$, so that we can use the Sign' algorithm that is compatible with the Gen' used to sample the key.[32] On the other hand, we generate a fresh $r_{\mathcal{S}}$ for verification, since including $r_{\mathcal{S}}$ in the inputs of Verify implies publishing it, which we cannot do. Since all the inputs and outputs to Verify' are visible to the environment in the ideal experiment involving $\mathcal{F}_{\mathsf{sig}}$, the behavior of Verify' cannot depend upon $r_{\mathcal{S}}$ (except insofar as its *public* inputs do).

In order to prove Theorem 3.7, we must first establish that $\Sigma_{\mathcal{S}}$ is linear time per Definition 2.5. Because $\mathcal{S}$ is polynomial-time and it receives a fixed-size input when it is invoked by $\mathcal{S}_{\texttt{algs}}$ with a particular security parameter, there must exist some specific polynomial that upper bounds the size of the description of Σ'. There must similarly exist some specific polynomial bound on the sizes

[31] Note that this is possible because Σ' is never revealed to the environment in the ideal experiment involving $\mathcal{F}_{\mathsf{sig}}$

[32] Note that $\mathcal{S}$ could (for example) use its random tape to generate obfuscated algorithms that use encrypted keys sk. This would mean that secret keys sk are incompatible between algorithms generated by evaluations of $\mathcal{S}_{\texttt{algs}}$ on differing random tapes. Because sk is never revealed to the environment, it would not be able to use such behavior to distinguish.

of the keys pk and sk' produced by Gen'.[33] We let $p(\kappa)$ be a specific polynomial bound on both. Since the time limits we place on Sign' and Verify' are linear in $p(\kappa)$, $\Sigma_{\mathcal{S}}$ satisfies the definition.[34] We will prove below that this gives Sign' and Verify' sufficient time to run, with all but negligible probability.

Next we must prove that the real world experiment involving $\pi_{\Sigma_{\mathcal{S}}}$ is indistinguishable from the real world experiment involving π_{sig} in the presence of of an adversary statically corrupting any number of participants. Formally, for dummy adversary $\mathcal{D}$,[35]

$$\left\{\textsc{Real}_{\pi_{\mathsf{sig}},\mathcal{D},\mathcal{Z}}(\kappa, z)\right\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*} \approx_{\mathrm{c}} \left\{\textsc{Real}_{\pi_{\Sigma_{\mathcal{S}}},\mathcal{D},\mathcal{Z}}(\kappa, z)\right\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}.$$

Our proof proceeds via a sequence of hybrid experiments, which begins with

$$\mathcal{H}_0 = \left\{\textsc{Real}_{\pi_{\Sigma_{\mathcal{S}}},\mathcal{D},\mathcal{Z}}(\kappa, z)\right\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}$$

and gradually modifies the behavior of the honest parties until they run the protocol π_{sig}.

Hybrid $\mathcal{H}_1$. This experiment is like $\mathcal{H}_0$, except that the time limit is removed from all calls to Verify' in the code of honest parties.

In order to prove that $\mathcal{H}_1 \approx_{\mathrm{c}} \mathcal{H}_0$, consider $\mathcal{H}_0$ from the point of view of Verify': Verify samples Verify' under one r_σ and then invokes on a public key and signatures that originated from a different r_σ, and Verify' cannot distinguish this case from the ideal world experiment in which it is sampled by $\mathcal{S}$ and then invoked by $\mathcal{F}_{\mathsf{sig}}$ on an adversarial public key and signature (i.e. ones not produced by $\mathcal{F}_{\mathsf{sig}}$). If Verify' times out in $\mathcal{H}_0$, then it must take more than

$$\begin{aligned} T(\kappa)\cdot(|m|+|\sigma|+1) &= p(\kappa)(p(\kappa)+1)(|m|+|\sigma|+1) \\ &\geq |\mathsf{Verify}|\cdot(|\mathsf{pk}|+1)(|m|+|\sigma|+1) \\ &\geq |\mathsf{Verify}|\cdot(|\mathsf{pk}|+|m|+|\sigma|+1) \end{aligned}$$

computational steps, which imples that $\mathcal{F}_{\mathsf{sig}}$ would change to random mode in the ideal world. By Lemma 3.9, we know this has negligible probability, and it follows that $\mathcal{H}_1 \approx_{\mathrm{c}} \mathcal{H}_0$.

Hybrid $\mathcal{H}_2$. This experiment is like $\mathcal{H}_1$, except that the time limit is removed from all calls to Sign' in the code of honest parties. Using a similar argument to the one we used to show that $\mathcal{H}_1 \approx_{\mathrm{c}} \mathcal{H}_0$, we can show that $\mathcal{H}_2 \approx_{\mathrm{c}} \mathcal{H}_1$.

Hybrid $\mathcal{H}_3$. Now, we run a second experiment in parallel with the first one, which contains an instance of π_{sig}. This experiment has no environment or adversary per se, and cannot be accessed by $\mathcal{Z}$ or $\mathcal{D}$ from the primary experiment. Let q_{G} be an upper bound on the number of `keygen` instructions issued in the primary experiment, and let q_{V} be an upper bound on the number of `verify` instructions issued. Since $\mathcal{Z}$ is PPT, both bounds are polynomial in κ. For every $i \in [q_{\mathsf{G}}]$, an honest party $P_{\mathsf{S}i}$ is created in the secondary experiment, and for every $i \in [q_{\mathsf{V}}]$, an honest party $P_{\mathsf{V}i}$ is created.

[33] If Gen' is not PPT, then $\mathcal{S}$ must cause $\mathcal{F}_{\mathsf{sig}}$ to enter random mode with nonnegligible probability, contradicting Lemma 3.9.

[34] Note that we do not limit Gen', since Definition 2.5 does not limit the runtime of Gen. We must still ensure that Gen is PPT, but this holds if Gen' is PPT.

[35] Per Claim 11 of Canetti [Can20], indistinguishability in the presence of $\mathcal{D}$ implies indistinguishability in the presence of any PPT adversary.

In addition, $\mathcal{H}_3$ includes a *router machine* $\mathcal{R}$, which has the following behavior: when it receives the i^{th} message of the form $(\texttt{keygen}, \mathsf{sid})$, it sends $(\texttt{init}, \mathsf{sid})$ and then $(\texttt{keygen}, \mathsf{sid})$ to the *input tape* of $P_{\mathsf{S}i}$.[36] When it receives the response $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk})$ from $P_{\mathsf{S}i}$, it forwards this message to P, where P is the party in the primary protocol who sent $\texttt{keygen}$. Thereafter, when P sends any message of the form $(\texttt{sign}, \mathsf{sid}, \mathsf{pk}, *)$ to $\mathcal{R}$, $\mathcal{R}$ forwards this message to the input tape of $P_{\mathsf{S}i}$, and then forwards the response back to P. Similarly, when $\mathcal{R}$ receives the i^{th} message of the form $(\texttt{verify}, \mathsf{sid}, *, *, *)$, it sends $(\texttt{init}, \mathsf{sid})$ to the input tape of $P_{\mathsf{V}i}$, then forwards the $\texttt{verify}$ message to the input tape of $P_{\mathsf{V}i}$, and then forwards the response of $P_{\mathsf{V}i}$ back to P, where P is the party that sent the i^{th} $\texttt{verify}$. Note that in all cases, honest parties in π_{sig} return activation to the environment; therefore when $\mathcal{R}$ activates some $P_{\mathsf{S}i}$ or $P_{\mathsf{V}i}$, activation returns to $\mathcal{R}$ immediately.

Since no honest party in the primary experiment in $\mathcal{H}_3$ ever sends a message to $\mathcal{R}$, the secondary experiment is completely independent of the primary experiment, and neither $\mathcal{Z}$ nor $\mathcal{D}$ can access it. Thus we have $\mathcal{H}_3 = \mathcal{H}_2$.

Hybrid $\mathcal{H}_4$. Like $\mathcal{H}_3$, this experiment includes a second parallel experiment involving π_{sig} and $q_{\mathsf{G}} + q_{\mathsf{V}}$ honest parties, and $\mathcal{R}$ taking the role of the environment. Whenever an honest party in the primary experiment in $\mathcal{H}_3$ invokes $\mathsf{Verify}'(\mathsf{pk}, m, \sigma)$, we replace its code in $\mathcal{H}_4$ with code that instead sends $(\texttt{verify}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ to $\mathcal{R}$, receives $(\texttt{verified}, \mathsf{sid}, \mathsf{pk}, m, \sigma, b)$ in response, and behaves as though b were the output of Verify'. Like $\mathcal{H}_0$, $\mathcal{H}_3$ is (nearly) identical to the ideal world experiment from the point of view of any particular Verify' call.[37] In $\mathcal{H}_4$, we replace each of these calls with an invocation of the corresponding interface of π_{sig} on a party that is never used for any other purpose.[38] Thus, our assumption that π_{sig} UC-realizes $\mathcal{F}_{\mathsf{sig}}$ implies that $\mathcal{H}_4 \approx_{\mathrm{c}} \mathcal{H}_3$.

Hybrid $\mathcal{H}_5$. In this experiment, we make a change similar to the one we made in $\mathcal{H}_4$. Whenever an honest party P in the primary experiment in $\mathcal{H}_4$ invokes $\mathsf{Gen}'()$ or $\mathsf{Sign}'(\mathsf{sk}', m)$, we replace its code in $\mathcal{H}_5$ with code that instead sends $(\texttt{keygen}, \mathsf{sid})$ or $(\texttt{sign}, \mathsf{sid}, \mathsf{pk}, m)$ to $\mathcal{R}$, respectively.[39] Note that P does *not* receive sk' from $\mathcal{R}$ in response to $\texttt{keygen}$, but this is not a problem, because it used sk' only to invoke Sign' and never revealed it to any other entity. Note that since $r_{\mathcal{S}}$ and $\mathcal{S}_{\texttt{algs}}$ are not used in $\mathcal{H}_5$, we can remove them. $\mathcal{H}_4$ is (nearly) identical to the ideal world experiment from the point of view of any particular sequence of one Gen' call and the Sign' calls that use the resulting sk'.[40] In $\mathcal{H}_5$, we replace each of these call sequences with a sequence of invocations of the corresponding interfaces of π_{sig} on a party that is used for nothing else.[38] As above, our assumption that π_{sig} UC-realizes $\mathcal{F}_{\mathsf{sig}}$ implies that $\mathcal{H}_5 \approx_{\mathrm{c}} \mathcal{H}_4$.

Hybrid $\mathcal{H}_6$. In $\mathcal{H}_6$, we replace the π_{sig} protocol instance in the secondary experiment of $\mathcal{H}_5$ with an ideal protocol invoking $\mathcal{F}_{\mathsf{sig}}$. We also add a new copy of $\mathcal{S}$ to the secondary experiment. Since there is no environment per se in the secondary experiment, $\mathcal{S}$ is activated only by $\mathcal{F}_{\mathsf{sig}}$ with the $\texttt{init}$ message. The quantifier order of UC insists that $\mathcal{S}$ must work for every PPT $\mathcal{Z}$ that expects to interact with the dummy adversary $\mathcal{D}$, including an $\mathcal{Z}$ that does not activate $\mathcal{S}$ before $\mathcal{F}_{\mathsf{sig}}$ does,

[36]The input tape is usually reserved for use only by the environment. Therefore, $\mathcal{R}$ is essentially taking the role of the environment in the secondary experiment.

[37]The two are nearly and not exactly identical because $\mathcal{H}_3$ does not limit the running time of Verify' and switch to random mode if that running time is exceeded. We established with Lemma 3.9 that this sequence of events happens with negligible probability.

[38] This prevents distinguishing based upon state preserved by π_{sig} between invocations.

[39]The sk' to pk mapping is performed in the obvious way: if in $\mathcal{H}_4$ Gen' produced an sk' that is later used with Sign', then the pk produced by the corresponding $\texttt{keygen}$ is used with the corresponding $\texttt{sign}$.

[40]Again, Lemma 3.9 ensures the difference between $\mathcal{H}_4$ and the ideal world is negligible from the point of view of any particular call sequence.

and never gives $\mathcal{S}$ the opportunity to corrupt any parties (see also footnote *a* in Algorithm 3.10). Thus $\mathcal{H}_6 \approx_c \mathcal{H}_5$ is implied by the fact that π_{sig} UC-realizes $\mathcal{F}_{\mathsf{sig}}$, and more specifically by the fact that for every PPT $\mathcal{Z}$,

$$\left\{\textsc{Real}_{\pi_{\mathsf{sig}},\mathcal{D},\mathcal{Z}}(\kappa, z)\right\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*} \approx_c \left\{\textsc{Ideal}_{\mathcal{F}_{\mathsf{sig}},\mathcal{S},\mathcal{Z}}(\kappa, z)\right\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}$$

Hybrid $\mathcal{H}_7$. This experiment changes the behavior of $\mathcal{R}$ and the parties in the secondary experiment. Whereas there were $q_{\mathsf{G}} + q_{\mathsf{V}}$ parties in the secondary experiment in $\mathcal{H}_6$, $\mathcal{H}_7$ includes exactly one party P'_i in the secondary experiment for each i^{th} honest party P_i in the primary expiriment. When $\mathcal{R}$ receives any message from P_i, it forwards it to P'_i, and vice versa, and before forwarding the *first* message, P'_i sends $(\texttt{init}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{sig}}$. In other words, if some party P_i in the primary experiment caused the activation of some set of parties in the secondary experiment in $\mathcal{H}_6$, then that set is replaced with a single party P'_i that receives the same commands in $\mathcal{H}_7$. Recall that the secondary-experiment parties in both hybrids simply forward their commands to $\mathcal{F}_{\mathsf{sig}}$, adding `init` commands beforehand if necessary. By inspection, we can see that the behavior of $\mathcal{F}_{\mathsf{sig}}$ does not depend upon who invokes it or how many commands each party invokes, so long as parties only attempt to sign under their own public keys (which is true in both $\mathcal{H}_7$ and $\mathcal{H}_6$); the state is only preserved between $\mathcal{F}_{\mathsf{sig}}$'s internal calls to the algorithms of Σ' via the secret keys. Therefore, $\mathcal{H}_7 = \mathcal{H}_6$.

Hybrid $\mathcal{H}_8$. This experiment simplifies the structure of $\mathcal{H}_7$. We observe that in $\mathcal{H}_7$, each honest party P_i in the primary experiment essentially just forwarded its inputs to $\mathcal{R}$ (after initialization), and forwarded $\mathcal{R}$'s responses back to $\mathcal{Z}$ as output. $\mathcal{R}$ in turn simply forwarded messages from P_i to P'_i and vice versa, and P'_i forwarded messages from $\mathcal{R}$ to $\mathcal{F}_{\mathsf{sig}}$ and vice versa (initializing $\mathcal{F}_{\mathsf{sig}}$ when necessary). In $\mathcal{H}_8$, P_i simply sends its inputs (including `init` commands) directly to $\mathcal{F}_{\mathsf{sig}}$, and outputs $\mathcal{F}_{\mathsf{sig}}$'s responses. In other words, the two parallel experiments are fully merged, and the honest parties in the primary (now only) experiment are essentially dummy parties for $\mathcal{F}_{\mathsf{sig}}$. Since the sequence of commands received by $\mathcal{F}_{\mathsf{sig}}$ is exactly the same in both hybrids, and the only outputs delivered by honest parties to $\mathcal{Z}$ in the primary experiment are those produced by $\mathcal{F}_{\mathsf{sig}}$ in both hybrids, we have $\mathcal{H}_8 = \mathcal{H}_7$.

Hybrid $\mathcal{H}_9$. Finally, this experiment is the real-world experiment involving π_{sig}; i.e.,

$$\mathcal{H}_9 = \left\{\textsc{Real}_{\pi_{\mathsf{sig}},\mathcal{D},\mathcal{Z}}(\kappa, z)\right\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}.$$

Notice that in $\mathcal{H}_8$, the joint view of the honest parties, $\mathcal{F}_{\mathsf{sig}}$, and $\mathcal{S}$ was indistinguishable from their view in an instance of

$$\left\{\textsc{Ideal}_{\mathcal{F}_{\mathsf{sig}},\mathcal{S},\mathcal{Z}}(\kappa, z)\right\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}$$

in which no parties were corrupted.[41] Therefore $\mathcal{H}_9 \approx_c \mathcal{H}_8$ is implied by the fact that

$$\left\{\textsc{Real}_{\pi_{\mathsf{sig}},\mathcal{D},\mathcal{Z}}(\kappa, z)\right\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*} \approx_c \left\{\textsc{Ideal}_{\mathcal{F}_{\mathsf{sig}},\mathcal{S},\mathcal{Z}}(\kappa, z)\right\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}$$

and Theorem 3.7 follows by transitivity. □

[41]The corrupted parties in $\mathcal{H}_8$, which nominally were running $\pi_{\Sigma_{\mathcal{S}}}$ and actually were doing whatever $\mathcal{A}$ instructed them to do, are simply part of the environment from this perspective.

4 A Modular Analysis of the Dolev-Strong Broadcast

In this section, we use the signature functionality from Section 3 to provide a modular analysis of the broadcast protocol of Dolev-Strong [DS83]. In Section 4.1 we review a model for synchronous protocols in UC, and in Section 4.2 we present a synchronous broadcast functionality $\mathcal{F}_{\mathsf{bc}}$. Then, in Section 4.3, we present the protocol of Dolev and Strong in the $\mathcal{F}_{\mathsf{sig}}$-hybrid model, and in Section 4.4 we prove that the protocol UC-realizes $\mathcal{F}_{\mathsf{bc}}$.

4.1 Modeling Synchronous Protocols in UC

In Section 3, we used the plain UC model to describe $\mathcal{F}_{\mathsf{sig}}$ and to prove its relationship with property-based signatures in the presence of a PPT environment. Dolev-Strong, however, requires synchrony, which is not captured by the standard UC framework. Therefore, in this section, we use the model of Katz et al. [KMTZ13], which builds upon UC in a compatible way and gives us a means to formally reason about synchrony. More specifically, we consider n parties $P_1, \ldots, P_n$ and a computationally *unbounded* adversary that adaptively corrupts up to t parties during the protocol execution. The Katz et al. model [KMTZ13] is a hybrid model wherein parties in sychronous protocols have access to a simple clock functionality $\mathcal{F}_{\mathsf{clock}}$.

$\mathcal{F}_{\mathsf{clock}}$ keeps an indicator bit. Once *all honest parties* request that the functionality switch this bit, it does so. After an honest party completes its operations for some round of a synchronous protocol, it requests that the bit be switched, indicating that it is ready to proceed to the next round of the protocol. When all honest parties are ready, the bit flips, signaling to them that the next round has begun. All communication between honest parties is performed via secure channels with bounded delay: specifically, a party can request that the channel deliver messages that were transmitted along it, and the adversary is allowed to delay message delivery by a bounded and a-priori known number of delivery requests. In other words, once the sender has transmitted a message, the model guarantees that the message will be delivered within a known number of activations of the receiver. For simplicity, we assume that every message is delivered within a single fetch request. We give a more detailed overview of the Katz et al. framework [KMTZ13] in Appendix B.

Canonical Synchronous Functionalities. Cohen et al. [CCGZ16] extended the framework of Katz et al. [KMTZ13] to capture protocols with probabilistic termination, i.e., protocols without a fixed output round and without simultaneous termination. Although the protocol of Dolev and Strong [DS83] has a deterministic and fixed number of rounds, some of the techniques from Cohen et al. simplify our presentation, and they allow us to capture additional protocols with expected-constant round complexity, such as the protocols of Katz and Koo [KK06] and Garay et al. [GKKO07].

The main idea behind Cohen et al.'s technique for modeling probabilistic termination is to separate the functionality to be computed from the round complexity that is required for the computation. The atomic building block of Cohen et al. is a functionality template called a *canonical synchronous functionality* (CSF), which is a simple two-round functionality with explicit (one-round) input and (one-round) output phases. The CSF functionality has two parameters: (1) a (possibly randomized) function f that receives $n + 1$ inputs (n inputs from the parties and one additional auxiliary input from the adversary[42]) and (2) a leakage function l that determines what

[42] The auxiliary input from the adversary is required in settings where the task to be computed is not a function per se, but it can be represented as a function if the adversary is allowed to provide a tie-breaker value as an auxiliary input. Consider Byzantine agreement, for example, wherein the output is indeterminate if the parties do

information about the input values is leaked to the adversary. The functionality proceeds in two rounds: In the first (input) round, all the parties hand $\mathcal{F}_{\mathsf{csf}}$ their input values. Whenever an input is submitted to $\mathcal{F}_{\mathsf{csf}}$, the adversary is given the image of this input under the leakage function; the adversary can use this leakage to decide which parties to corrupt and which input values to use for corrupted parties. At this point, the adversary also provides its own input, if one is required by f. In the second (output) round, each party receives its output.

Since the focus of this work is broadcast protocols, we define the broadcast functionality by explicitly incorporating the functions f and l into the CSF template. To capture multiple instances of broadcast over the same PKI, we will define broadcast as a reactive CSF functionality (similarly to the prior work of Cohen et al. [CCGZ17]).

Round-Extending Wrappers. While a CSF describes the essence of an ideal computation, it does not define the round complexity of a protocol realizing that realizes an ideal computation. Cohen et al. [CCGZ16] captured the round structure of a protocol using *round-extending wrappers.* Such wrappers are parametrized by a distribution $\mathcal{D}$ that may depend on a specific protocol implementing the functionality. A wrapper samples a round $\rho_{\mathsf{term}} \leftarrow \mathcal{D}$ by which all parties are guaranteed to receive their outputs. Two wrappers are defined by Cohen et al.: The first, denoted $\mathcal{W}_{\mathsf{strict}}$, ensures in a strict manner that all honest parties terminate together in round ρ_{term}; the second, denoted $\mathcal{W}_{\mathsf{flex}}$, is more flexible and allows the adversary to deliver outputs to individual parties at any time before round ρ_{term}.[43] To model the Dolev-Strong protocol [DS83] we will only require the simpler $\mathcal{W}_{\mathsf{strict}}$ wrapper, with a deterministic ρ_{term} value, which we hardcode for simplicity into a specialized $\mathcal{W}_{\mathsf{DS}}$ wrapper. We formalize this in the next section as Functionality 4.2.

4.2 The Broadcast Functionality

In a broadcast protocol, a designated sender distributes its message to all recipients in a way that ensures *agreement*, *validity*, and *termination*. As discussed in Section 4.1, we follow the approach of Cohen et al. [CCGZ16] and model broadcast as a *canonical synchronous functionality*, in which the function to compute simply copies the sender's input to the outputs of all receivers,[44] and the leakage function outputs the sender's input (i.e., the entire message). We formalize this behavior below, as Functionality 4.1.

With the goal of MPC protocols in mind, we wish to capture multiple instances of broadcast with a known set of parties using the same PKI. Therefore, we model broadcast as a reactive functionality in which the first phase is a registration phase where every party registers to the functionality. This phase corresponds to the offline phase of establishing the PKI in broadcast protocols; a party that does not register cannot participate in future broadcasts. For each party P_i, the functionality maintains a boolean flag active_i indicating whether P_i is registered.

After the registration phase, each reactively-invoked broadcast phase has a unique sub-session id ssid that specifies the sender P_s and the set of recipients $\mathcal{P} \subseteq \{P_1, \ldots, P_n\}$ (all of whom must be registered). In the input round, the sender sends its message (denoted m_{out}) and the others provide no input (formally, they send an empty string). We note that a *corrupted* party can provide its input multiple times during the input round, which implies that the adversary can adaptively corrupt an honest sender based upon the initial value of m_{out}, and then replace m_{out} with a different message. In the output round, each recipient will receive the most recent m_{out}.

not pre-agree.

[43]The flexible wrapper is useful for modeling expected-constant-round protocols.

[44]There is no need for the auxiliary input from the adversary; see Footnote 42.

Once a sender provides its input to some broadcast phase, the functionality sends leakage to the ideal-world adversary. We define this leakage to be the sender's entire input. This corresponds to the traditional, property-based notion of broadcast. We do not capture stronger security notions that naturally arise in simulation-based definitions of broadcast [HZ10, GKKZ11, CGZ23]. For example, we do not attempt to guarantee *corruption fairness* [CGZ23], which insists that the adversary cannot adaptively corrupt the sender and then replace its message, based upon the initial message that it sent when it was honest. To model this stronger notion of broadcast, the leakage function must be adjusted such that it reveals only the length of the sender's message to the adversary, rather than its contents. Although corruption-unfair broadcast is weaker, it is still sufficient for many MPC protocols, e.g., [GMW87, CLOS02]; furthermore, it can be enhanced to achieve corruption-fairness using atomic multisend channels and equivocal commitments [GKKZ11], or using time-lock puzzles in the programmable random oracle model [CGZ23].

Functionality 4.1. $\mathcal{F}_{\mathsf{bc}}$ (The Canonical Synchronous Functionality for Broadcast)

This functionality interacts with an ideal adversary $\mathcal{S}$ and parties $P_1, \ldots, P_n$. It has some memory associated with each unique session (distinguished by session ID sid), and in that memory it keeps the boolean values $\mathsf{active}_1, \ldots, \mathsf{active}_n$, which assume the value 0 initially and indicate whether each respective P_i has registered. It also associates a value m_{out} with every *sub*session (distinguished by the subsession ID ssid), which assumes the value $\perp$ initially. This functionality achieves synchrony through the framework of Katz et al. [KMTZ13], i.e., it only proceeds from one round to the next when all honest parties have indicated that it should do so. An honest party can be adaptively corrupted at any point, after which, depending on when the party is corrupted, the ideal adversary may send messages on its behalf and potentially overwrite previously stored values. In every round, the functionality ignores all messages other than the ones we explicitly specify that it can receive.

Setup.

1. **Input Round:** Upon receiving $(\texttt{init}, \mathsf{sid})$ from some party P_i, set $\mathsf{active}_i := 1$ and send $(\texttt{init}, \mathsf{sid}, P_i)$ to $\mathcal{S}$.
2. **Output Round:** Upon receiving $(\texttt{fetch-output}, \mathsf{sid})$ from some P_i, send $(\mathsf{output}, \mathsf{sid}, (\mathsf{active}_1, \ldots, \mathsf{active}_n))$ to P_i.

Broadcast.

3. **Input Round:** Upon receiving $(\texttt{input}, \mathsf{sid}, \mathsf{ssid}, v)$ from some party P_s where ssid is of the form $\mathsf{ssid}' \| P_s \| \mathcal{P}$: if there exists a $P_j \in \mathcal{P} \cup \{P_s\}$ such that $\mathsf{active}_j = 0$, then ignore the message. Otherwise, set $m_{\mathsf{out}} := v$ and send $(\texttt{input}, \mathsf{sid}, \mathsf{ssid}, v, P_i)$ to $\mathcal{S}$.
4. **Output Round:** Upon receiving $(\texttt{fetch-output}, \mathsf{sid}, \mathsf{ssid})$ from some party P_i, send $(\mathsf{output}, \mathsf{sid}, \mathsf{ssid}, m_{\mathsf{out}})$ to P_i.

Next, we define the *strict-termination wrapper functionality* corresponding to the realization of $\mathcal{F}_{\mathsf{bc}}$ with the Dolev-Strong protocol [DS83]. Recall that the registration phase of $\mathcal{F}_{\mathsf{bc}}$ corresponds to the setup phase of establishing the PKI. The Dolev-Strong protocol assumes that the PKI has already been established, i.e., it works in the $\mathcal{F}_{\mathsf{pki}}$-hybrid model, where $\mathcal{F}_{\mathsf{pki}}$ (provided in Section 4.3) is an ideal functionality that establishes a public-key infrastructure. Thus, we set the output round for the setup phase to 2 (one round for providing inputs to $\mathcal{F}_{\mathsf{pki}}$ and one round for receiving output). In every other phase, i.e., a broadcast phase for sub-session id of the form $\mathsf{ssid}' \| P_s \| \mathcal{P}$, the output round is specified to be $\rho_{\mathsf{term}} := |\mathcal{P}| + 1$. This ensures security in the presence of an adversary corrupting any subset of $\mathcal{P}$.

Functionality 4.2. $\mathcal{W}_{\mathsf{DS}}(\mathcal{F}_{\mathsf{bc}})$ (The Strict Wrapper for Dolev-Strong [CCGZ16])

The wrapper functionality $\mathcal{W}_{\mathsf{DS}}(\mathcal{F}_{\mathsf{bc}})$ interacts with an ideal adversary $\mathcal{S}$ and parties $P_1, \ldots, P_n$, and it runs a copy of $\mathcal{F}_{\mathsf{bc}}$ internally.

Setup.

1. In round $\rho = 1$: Forward $(\mathtt{init}, \mathsf{sid})$ messages from each party P_i to $\mathcal{F}_{\mathsf{bc}}$. In addition, forward all messages between the back-door tape of $\mathcal{F}_{\mathsf{bc}}$ and the adversary.
2. In round $\rho = 2$: Forward $(\mathtt{fetch\text{-}output}, \mathsf{sid})$ messages from each party P_i to $\mathcal{F}_{\mathsf{bc}}$ and the response $(\mathsf{output}, \mathsf{sid}, *)$ to P_i.

Broadcast.

3. Upon receiving the *first* message of the form $(\mathtt{input}, \mathsf{sid}, \mathsf{ssid}, *)$ for a fresh ssid, parse ssid as $\mathsf{ssid}' \| P_s \| \mathcal{P}$, set the output round $\rho_{\mathsf{term}} := |\mathcal{P}| + 1$, and send $(\mathsf{output\text{-}round}, \mathsf{sid}, \mathsf{ssid}, \rho_{\mathsf{term}})$ to the adversary. Continue processing this message using the rules enumerated below.
4. At all times, forward $(\mathtt{input}, \mathsf{sid}, \mathsf{ssid}, *)$ messages from corrupted parties to $\mathcal{F}_{\mathsf{bc}}$.
5. In round $\rho = 1$: Forward $(\mathtt{input}, \mathsf{sid}, \mathsf{ssid}, *)$ messages from each party $P_i \in \mathcal{P}$ to $\mathcal{F}_{\mathsf{bc}}$. In addition, forward all messages between the back-door tape of $\mathcal{F}_{\mathsf{bc}}$ and the adversary.
6. In rounds $\rho > 1$: Upon receiving a message $(\mathtt{fetch\text{-}output}, \mathsf{sid}, \mathsf{ssid})$ from some party $P_i \in \mathcal{P}$:
   - If $\rho = \rho_{\mathsf{term}}$, forward the message to $\mathcal{F}_{\mathsf{bc}}$, and the response $(\mathsf{output}, \mathsf{sid}, \mathsf{ssid}, *)$ to P_i.
   - Otherwise, send $(\mathtt{fetch\text{-}output}, \mathsf{sid}, \mathsf{ssid}, P_i)$ to the adversary.

4.3 The Dolev-Strong Broadcast Protocol

Signatures on their own are not sufficient to overcome the impossibility of broadcasts when $t \geq n/3$ [PSL80, FLM86]; as we have alluded previously, we also need a PKI functionality. Much like our broadcast functionality, our PKI functionality is a CSF (see Section 4.1).

Functionality 4.3. $\mathcal{F}_{\mathsf{pki}}$ (The PKI Functionality)

This functionality interacts with an ideal adversary $\mathcal{S}$ and parties $P_1, \ldots, P_n$. It has some memory associated with each unique session (distinguished by session ID sid), and in that memory it keeps the boolean values $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, which assume the value $\perp$ initially and store the values that each respective P_i has registered. This functionality achieves synchrony through the framework of Katz et al. [KMTZ13], i.e., it only proceeds from one round to the next when all honest parties have indicated that it should do so. An honest party can be adaptively corrupted at any point, after which, depending on when the party is corrupted, the ideal adversary may send messages on its behalf and potentially overwrite previously stored values. In every round, the functionality ignores all messages other than the ones we explicitly specify that it can receive.

Registration.

1. **Input Round:** Upon receiving $(\texttt{input}, \mathsf{sid}, m)$ from some party P_i, set $\mathsf{pk}_i := m$ and send $(\texttt{input}, \mathsf{sid}, P_i, m)$ to $\mathcal{S}$.
2. **Output Round:** Upon receiving $(\texttt{fetch-output}, \mathsf{sid})$ from some party P_i, send $(\mathsf{output}, \mathsf{sid}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n))$ to P_i.

Now we are finally ready to describe the Dolev-Strong protocol [DS83] in the $(\mathcal{F}_{\mathsf{sig}}, \mathcal{F}_{\mathsf{pki}})$-hybrid model. We refer the reader to Section 1.3 for a high-level overview of our design choices.

Protocol 4.4. π_{DS} (Dolev-Strong Broadcast in the $(\mathcal{F}_{\mathsf{sig}}, \mathcal{F}_{\mathsf{pki}})$-Hybrid Model)

This protocol is run among n parties $P_1, \ldots, P_n$ in the $(\mathcal{F}_{\mathsf{sig}}, \mathcal{F}_{\mathsf{pki}})$-hybrid model. During the protocol, all parties ignore messages that are not explicitly specified.

Setup. Upon receiving $(\texttt{init}, \mathsf{sid})$ from the environment, party P_i proceeds as follows:

1. **Round 1**: P_i sends $(\texttt{init}, \mathsf{sid})$ followed by $(\texttt{keygen}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{sig}}$ and receives $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk}_i)$ in response. Then it sends $(\texttt{input}, \mathsf{sid}, \mathsf{pk}_i)$ to $\mathcal{F}_{\mathsf{pki}}$.
2. **Round 2**: P_i sends $(\texttt{fetch-output}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{pki}}$ and receives $(\mathsf{output}, \mathsf{sid}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n))$ in response. For every $j \in [n]$, P_i internally sets $\mathsf{active}_j := 0$ if $\mathsf{pk}_j = \bot$, and $\mathsf{active}_j := 1$ otherwise. P_i outputs $(\mathsf{output}, \mathsf{sid}, (\mathsf{active}_1, \ldots, \mathsf{active}_n))$ to $\mathcal{Z}$.

Broadcast. Upon receiving $(\texttt{input}, \mathsf{sid}, \mathsf{ssid}, v)$ from the environment such that $\mathsf{ssid} = \mathsf{ssid}' \| P_s \| \mathcal{P}$, party P_i proceeds as follows:

3. **Round 1**:
   (a) P_i verifies that $\mathcal{P} \subseteq \{P_1, \ldots, P_n\}$, that $P_i \in \mathcal{P}$, and that for every $P_j \in \mathcal{P}$, $\mathsf{active}_j = 1$. If these conditions hold, then P_i initializes a set $\mathcal{V}_i^1 := \emptyset$; otherwise, it ignores the environment's message. The sets $\mathcal{V}_i^\rho$ correspond to the possible eventual outputs of party P_i as of round ρ.
   (b) If $P_i = P_s$, then P_i sets $m := v$, sends $(\texttt{sign}, \mathsf{sid}, \mathsf{pk}_i, (\mathsf{sid}, \mathsf{ssid}, i, m))$ to $\mathcal{F}_{\mathsf{sig}}$, and receives $(\texttt{signature}, \mathsf{sid}, \mathsf{pk}_i, (\mathsf{sid}, \mathsf{ssid}, i, m), \sigma_i)$ in response. Next, P_i sends $(\mathsf{sid}, \mathsf{ssid}, m, (i, \sigma_i))$ to all other parties in $\mathcal{P}$.
   (c) If $P_i \neq P_s$, then it ignores v.[a]
4. **Round ρ, for $\rho \in \{2, \ldots, |\mathcal{P}|\}$:** P_i initializes $\mathcal{V}_i^\rho := \mathcal{V}_i^{\rho-1}$ and receives between 0 and $|\mathcal{P}| - 1$ messages of the form $(\mathsf{sid}, \mathsf{ssid}, \tilde{m}, (j_1, \sigma_{j_1}), \ldots, (j_{\rho-1}, \sigma_{j_{\rho-1}}))$.[b] It processes every such message as follows:
   (a) P_i ignores the message[c] if $\tilde{m} \in \mathcal{V}_i^\rho$, or if $|\mathcal{V}_i^\rho| > 1$, or if the indices $j_1, \ldots, j_{\rho-1}$ are not all distinct, or if $i \in \{j_1, \ldots, j_{\rho-1}\}$, or if $j_1 \neq s$, where P_s is the designated sender.

(b) For $k \in [\rho]$, let $\mathring{m}_k := \tilde{m} \| j_1 \| \sigma_{j_1} \| \ldots \| j_{k-1} \| \sigma_{j_{k-1}}$. For $k \in [\rho-1]$, P_i sends $(\texttt{verify}, \mathsf{sid}, \mathsf{pk}_{j_k}, (\mathsf{sid}, \mathsf{ssid}, j_k, \mathring{m}_k), \sigma_{j_k})$ to $\mathcal{F}_{\mathsf{sig}}$ and receives $(\texttt{verified}, \mathsf{sid}, \mathsf{pk}_{j_k}, (\mathsf{sid}, \mathsf{ssid}, j_k, \mathring{m}_k), \sigma_{j_k}, b_k)$ in response.

(c) If $\bigwedge_{k \in [\rho-1]} b_k = 1$,[d] then P_i sends $(\texttt{sign}, \mathsf{sid}, \mathsf{pk}_i, (\mathsf{sid}, \mathsf{ssid}, i, \mathring{m}_\rho))$ to $\mathcal{F}_{\mathsf{sig}}$, receives $(\texttt{signature}, \mathsf{sid}, \mathsf{pk}_i, (\mathsf{sid}, \mathsf{ssid}, i, \mathring{m}_\rho), \sigma_i)$ in response, and sends $(\mathsf{sid}, \mathsf{ssid}, \tilde{m}, (j_1, \sigma_{j_1}), \ldots, (j_{\rho-1}, \sigma_{j_{\rho-1}}), (i, \sigma_i))$ to every party $P \in \mathcal{P} \setminus \{P_{j_1}, \ldots, P_{j_{\rho-1}}, P_i\}$.

(d) If $\bigwedge_{k \in [\rho-1]} b_k = 1$,[d], then P_i updates $\mathcal{V}_i^\rho := \mathcal{V}_i^\rho \cup \{\tilde{m}\}$.

5. **Round $|\mathcal{P}|+1$**: If the sender P_s received $(\texttt{input}, \mathsf{sid}, \mathsf{ssid}, m)$ in Round 1, then it outputs $(\mathsf{output}, \mathsf{sid}, \mathsf{ssid}, m)$ to $\mathcal{Z}$; otherwise, it outputs $(\mathsf{output}, \mathsf{sid}, \mathsf{ssid}, \bot)$.

   Every other party P_i checks whether $\mathcal{V}_i^{|\mathcal{P}|} = \{m_i\}$ for some m_i. If so (i.e., $|\mathcal{V}_i^{|\mathcal{P}|}| = 1$) then P_i outputs $(\mathsf{output}, \mathsf{sid}, \mathsf{ssid}, m_i)$ to $\mathcal{Z}$; otherwise, it outputs $(\mathsf{output}, \mathsf{sid}, \mathsf{ssid}, \bot)$.

[a] The party who is not the sender does not have a meaningful input in this phase.
[b] The values of $\tilde{m}$, j_*, and σ_{j_*} may differ in each message.
[c] That is, is performs no further steps related to the message.
[d] That is, all signatures verified in Step 4b.

4.4 A Modular Proof of Security for Dolev-Strong

Theorem 4.5. *The protocol π_{DS}* perfectly *UC-realizes $\mathcal{W}_{\mathsf{DS}}(\mathcal{F}_{\mathsf{bc}})$ in the $(\mathcal{F}_{\mathsf{sig}}, \mathcal{F}_{\mathsf{pki}})$-hybrid model against a (possibly unbounded) malicious adversary that adaptively corrupts any set of parties.*

Proof. Let $\mathcal{A}$ be any computationally unbounded malicious adversary that adaptively corrupts any group of participants in the protocol π_{DS}. We will construct a PPT ideal-process adversary $\mathcal{S}_{\mathsf{DS}}$ that runs $\mathcal{A}$ as a black-box subroutine[45] and interacts with the ideal functionality $\mathcal{W}_{\mathsf{DS}}(\mathcal{F}_{\mathsf{bc}})$, and we will prove that for every computationally unbounded environment $\mathcal{Z}$,

$$\left\{\textsc{Real}_{\pi_{\mathsf{DS}}, \mathcal{A}, \mathcal{Z}}(\kappa, z)\right\}_{\kappa \in \mathbb{N}, z \in \{0,1\}^*} = \left\{\textsc{Ideal}_{\mathcal{W}_{\mathsf{DS}}(\mathcal{F}_{\mathsf{bc}}), \mathcal{S}_{\mathsf{DS}}^{\mathcal{A}}, \mathcal{Z}}(\kappa, z)\right\}_{\kappa \in \mathbb{N}, z \in \{0,1\}^*}.$$

In the ideal-world experiment, we denote the dummy parties as $\tilde{P}$. We begin by specifying the simulator.

Simulator 4.6. $\mathcal{S}_{\mathsf{DS}}$ (Simulator for Dolev-Strong Broadcast)

$\mathcal{S}_{\mathsf{DS}}$ emulates an instance of the π_{DS} protocol internally, including n virtual parties $P_1, \ldots, P_n$ and the functionalities $\mathcal{F}_{\mathsf{sig}}$ and $\mathcal{F}_{\mathsf{pki}}$, which all behave exactly according to their specifications unless otherwise stated. In addition, $\mathcal{S}_{\mathsf{DS}}$ uses $\mathcal{A}$ as a black-box subroutine and emulates for it communication with the various protocol entitites in the virtual π_{DS} instance. In particular, the back door $\texttt{init}$ message by which $\mathcal{F}_{\mathsf{sig}}$ requests its algorithms is forwarded to $\mathcal{A}$, and if $\mathcal{A}$ does not immediately reply, then the emulated $\mathcal{F}_{\mathsf{sig}}$ goes into random mode as expected. All messages from $\mathcal{Z}$ to $\mathcal{S}_{\mathsf{DS}}$ are forwarded to $\mathcal{A}$, and all values written on $\mathcal{A}$'s output tape are forwarded to $\mathcal{Z}$ via $\mathcal{S}_{\mathsf{DS}}$'s output tape.

Corruption Requests. When the parties in π_{DS} behave honestly, they have no random choices, and therefore need no random tapes. Observe that whenever $\mathcal{W}_{\mathsf{DS}}(\mathcal{F}_{\mathsf{bc}})$ receives

[45] It is essential that $\mathcal{S}_{\mathsf{DS}}$ be PPT in order to ensure that the UC composition theorem holds in computationally-bounded contexts.

an input from an honest party, it forwards that input to $\mathcal{S}_{\mathsf{DS}}$. Consequently, the activation order and emulated input tapes of the virtual parties exactly match the equivalent tapes in the real world experiment. The functionalities are randomized, but are emulated using the same code they use in π_{DS}; together these facts imply that the communication and output tapes of the virtual parties are distributed identically to the tapes of the actual parties in the real world π_{DS} experiment. When $\mathcal{A}$ requests to corrupt some party P, $\mathcal{S}_{\mathsf{DS}}$ corrupts the corresponding dummy party $\tilde{P}$ in the ideal world experiment and transmits the internal state of the virtual P to $\mathcal{A}$. Additionally $\mathcal{S}_{\mathsf{DS}}$ emulates the adaptive corruption processes of $\mathcal{F}_{\mathsf{pki}}$ and $\mathcal{F}_{\mathsf{sig}}$ to $\mathcal{A}$, which is possible since $\mathcal{S}_{\mathsf{DS}}$ knows those functionalities' entire internal states. From that point onward, $\mathcal{S}_{\mathsf{DS}}$ ceases to emulate P, and expects $\mathcal{A}$ to receive and produce messages P's behalf.

Setup.

1. Upon receiving $(\mathtt{init}, \mathsf{sid}, \tilde{P}_i)$ from $\mathcal{W}_{\mathsf{DS}}(\mathcal{F}_{\mathsf{bc}})$ on its back door tape such that $\tilde{P}_i$ is honest, $\mathcal{S}_{\mathsf{DS}}$ places $(\mathtt{init}, \mathsf{sid})$ on the input tape of virtual P_i and activates it, emulating the operation of the protocol that results.
2. When $\mathcal{A}$ transmits $(\mathtt{input}, \mathsf{sid}, \mathsf{pk}_i)$ to $\mathcal{F}_{\mathsf{pki}}$ on behalf of a corrupted party P_i, $\mathcal{S}_{\mathsf{DS}}$ sends $(\mathtt{init}, \mathsf{sid})$ to $\mathcal{W}_{\mathsf{DS}}(\mathcal{F}_{\mathsf{bc}})$ on behalf of $\tilde{P}_i$, in addition to emulating the protocol π_{DS} towards $\mathcal{A}$.
3. When $\mathcal{A}$ transmits $(\mathtt{fetch\text{-}output}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{pki}}$ on behalf of a corrupted party P_i, $\mathcal{S}_{\mathsf{DS}}$ forwards this message to $\mathcal{W}_{\mathsf{DS}}(\mathcal{F}_{\mathsf{bc}})$ on behalf of $\tilde{P}_i$.
4. $\mathcal{S}_{\mathsf{DS}}$ releases the corrupted P_i's output $(\mathsf{output}, \mathsf{sid}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n))$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{pki}}$ *only* when it receives $(\mathsf{output}, \mathsf{sid}, (\mathsf{active}_1, \ldots, \mathsf{active}_n))$ from $\mathcal{W}_{\mathsf{DS}}(\mathcal{F}_{\mathsf{bc}})$ on behalf of $\tilde{P}_i$.

Broadcast.

5. **Round 1:** Upon receiving $(\mathtt{input}, \mathsf{sid}, \mathsf{ssid}, v, \tilde{P}_i)$ from $\mathcal{W}_{\mathsf{DS}}(\mathcal{F}_{\mathsf{bc}})$ on its back door tape such that $\tilde{P}_i$ is honest, $\mathcal{S}_{\mathsf{DS}}$ places $(\mathtt{input}, \mathsf{sid}, \mathsf{ssid}, v)$ on the input tape of virtual P_i and activates it, emulating the operation of the protocol that results.
6. **Round ρ, for $\rho \in \{2, \ldots, |\mathcal{P}|\}$:** The simulator does not communicate with $\mathcal{W}_{\mathsf{DS}}(\mathcal{F}_{\mathsf{bc}})$ in these rounds,[a] but continues to simulate the corresponding rounds of π_{DS} using the internally emulated honest parties and functionalities, which communicate with $\mathcal{A}$ (who acts on behalf of corrupt parties) and with one another.
7. **Round $|\mathcal{P}|$** : If the emulated sender P_s is corrupted by $\mathcal{A}$ before the end of round $|\mathcal{P}|$, and at least one party in $\mathcal{P}$ remains honest when round $|\mathcal{P}|$ ends, then $\mathcal{S}_{\mathsf{DS}}$ selects one honest party P_{i^*} arbitrarily among those that remain. If $\mathcal{V}_{i^*}^{|\mathcal{P}|} = \{\hat{m}\}$ for some message $\hat{m}$ (i.e., $|\mathcal{V}_{i^*}^{|\mathcal{P}|}| = 1$), then $\mathcal{S}_{\mathsf{DS}}$ sets $m' = \hat{m}$; otherwise it sets $m' = \perp$. Regardless, $\mathcal{S}_{\mathsf{DS}}$ sends $(\mathtt{input}, \mathsf{sid}, \mathsf{ssid}, m')$ to $\mathcal{W}_{\mathsf{DS}}(\mathcal{F}_{\mathsf{bc}})$ on behalf of the corrupt sender $\tilde{P}_s$. If the emulated sender is not corrupt, or no parties in $\mathcal{P}$ remain honest, then $\mathcal{S}_{\mathsf{DS}}$ sends nothing to $\mathcal{W}_{\mathsf{DS}}(\mathcal{F}_{\mathsf{bc}})$.
8. **Round $|\mathcal{P}|+1$:** $\mathcal{S}_{\mathsf{DS}}$ does not communicate with $\mathcal{W}_{\mathsf{DS}}(\mathcal{F}_{\mathsf{bc}})$ in this round, but continues to simulate the corresponding round of π_{DS} using the internally emulated honest parties and functionalities, which communicate with $\mathcal{A}$. Note that the emulated honest parties do not send an messages in this round (though they may be corrupted), and that any

dummy party $\tilde{P}$ that remains honest at the end of the round receives its ideal-world output directly from $\mathcal{W}_{\mathsf{DS}}(\mathcal{F}_{\mathsf{bc}})$.

[a]Except as specified in Step 7

Since $\mathcal{S}_{\mathsf{DS}}$ emulates π_{DS} internally by following the exact instructions of the honest parties, the joint view of $\mathcal{A}$ and $\mathcal{Z}$ must be distributed identically in the real and ideal worlds until the honest parties reveal their outputs to $\mathcal{Z}$ in round $|\mathcal{P}|+1$. It is therefore sufficient to prove that the outputs they produce in the ideal world are identical to their outputs in the real world. We divide the remainder of the proof into two cases: the first applies to the event that the sender remains honest at least until the end of round $|\mathcal{P}|$, and the second to the complementary event that the sender is corrupted before the end of round $|\mathcal{P}|$. Our theorem follows from the conjunction of Lemmas 4.7 and 4.8, which assert that the distribution of honest party outputs is identical between the real and ideal worlds in these two respective cases. □

Lemma 4.7. *If the sender remains honest until the end of round $|\mathcal{P}|$, then the outputs of honest dummy parties in the ideal experiment involving $\mathcal{W}_{\mathsf{DS}}(\mathcal{F}_{\mathsf{bc}})$ and $\mathcal{S}^{\mathcal{A}}_{\mathsf{DS}}$ are distributed identically to the outputs of honest parties in the real experiment involving π_{DS} and $\mathcal{A}$.*

Proof. If the (virtual or real) sender P_s is honest, then in both $\textsc{Real}_{\pi_{\mathsf{DS}},\mathcal{A},\mathcal{Z}}$ and $\textsc{Ideal}_{\mathcal{W}_{\mathsf{DS}}(\mathcal{F}_{\mathsf{bc}}),\mathcal{S}^{\mathcal{A}}_{\mathsf{DS}},\mathcal{Z}}$, P_s sends its signature σ_s on $(\mathsf{sid},\mathsf{ssid},s,m)$ to all the other participants in round 1. Then, in round 2, each honest party P_i receives it. No P_i can ever receive any $m' \neq m$ and σ'_s such that σ'_s verifies on $(\mathsf{sid},\mathsf{ssid},s,m')$ under the sender's public key, since this would imply either that σ'_s was the product of a signing request to $\mathcal{F}_{\mathsf{sig}}$ on the part of P_s (which cannot occur if P_s is honest) or that σ'_s is a forgery (which happens with a probability of exactly 0, per the specification of $\mathcal{F}_{\mathsf{sig}}$). Thus, at the end of round 2, $\mathcal{V}^2_i = \{m\}$ for each honest P_i. In round $\rho > 2$, we also have $\mathcal{V}^{\rho}_i = \{m\}$ for each honest P_i, via the same reasoning: no signature on $(\mathsf{sid},\mathsf{ssid},s,m')$ for $m' \neq m$ will ever verify under the sender's public key, because the sender will never request such a signature, and forgeries are impossible. Thus, in the last round, $V^{|\mathcal{P}|}_i = \{m\}$ and every honest P_i outputs m in π_{DS} in the real-world experiment. In the ideal world experiment, the output of the virtual honest parties is ignored, but the specification of $\mathcal{W}_{\mathsf{DS}}(\mathcal{F}_{\mathsf{bc}})$ guarantees that all honest dummy parties similarly output the same value m. □

Lemma 4.8. *If the sender is corrupted before the end of round $|\mathcal{P}|$, then the outputs of honest dummy parties in the ideal experiment involving $\mathcal{W}_{\mathsf{DS}}(\mathcal{F}_{\mathsf{bc}})$ and $\mathcal{S}^{\mathcal{A}}_{\mathsf{DS}}$ are distributed identically to the outputs of honest parties in the real experiment involving π_{DS} and $\mathcal{A}$.*

Before we prove Lemma 4.8, we prove a useful building-block lemma:

Lemma 4.9. *If P_i and P_j are both honest participants in π_{DS}, then either $\mathcal{V}^{|\mathcal{P}|}_i \subseteq \mathcal{V}^{|\mathcal{P}|}_j$ or $|\mathcal{V}^{|\mathcal{P}|}_j| > 1$.*

Proof of Lemma 4.9. The claim trivially holds when $\mathcal{V}^{|\mathcal{P}|}_i = \emptyset$. Consider when $|\mathcal{V}^{|\mathcal{P}|}_i| > 0$. Let $v \in \mathcal{V}^{|\mathcal{P}|}_i$ and let ρ be the round when P_i first received the properly authenticated value v and added it to $\mathcal{V}^{\rho}_i$. There are two mutually exclusive and jointly comprehensive cases for ρ that we need to consider:

1. $\boldsymbol{\rho < |\mathcal{P}|}$ **:** In this case, P_j will receive from P_i the value v together with a valid chain of signatures in round $\rho+1 \leq |\mathcal{P}|$. At this point, either $|\mathcal{V}^{\rho+1}_j| > 1$ *already*, or else P_j will add v to $\mathcal{V}^{\rho+1}_j$ at this point. Therefore, $v \in \mathcal{V}^{|\mathcal{P}|}_j$.

2. $\boldsymbol{\rho = |\mathcal{P}|}$: Because $|\mathcal{P}|$ is the first round in which P_i added v to $\mathcal{V}_i^\rho$, P_i *must* have received a signature chain containing the valid signatures of $|\mathcal{P}| - 1$ distinct participants on v. Since those are *all* of the participants other than P_i, P_j must be among them. Since an honest P_j only signs a value in some round ρ' if it also adds that value to $\mathcal{V}_j^{\rho'}$, it must be the case that $v \in \mathcal{V}_j^{\rho'}$ for some $\rho' < |\mathcal{P}|$, and therefore $v \in \mathcal{V}_j^{|\mathcal{P}|}$ already.

By the conjunction of these two cases, if $v \in \mathcal{V}_i^\rho$ for any v and ρ, then either $v \in \mathcal{V}_j^{|\mathcal{P}|}$ or $|\mathcal{V}_j^{|\mathcal{P}|}| > 1$. Therefore, either $\mathcal{V}_i^{|\mathcal{P}|} \subseteq \mathcal{V}_j^{|\mathcal{P}|}$ or $|\mathcal{V}_j^{|\mathcal{P}|}| > 1$. □

Proof of Lemma 4.8. The lemma trivially holds if all participants are corrupted by the last round, since in that case there are no honest participants to produce an output. Hereafter we consider the case that not all participants are corrupted.

Let P_{i^*} be the virtual honest party whose state the simulator $\mathcal{S}$ uses to determine the message m' that it sends to $\mathcal{W}_{\mathsf{DS}}(\mathcal{F}_{\mathsf{bc}})$ at the end of round $|\mathcal{P}|$ (in Step 7), whereafter m' becomes the output of *all* honest dummy parties in the ideal-world experiment. Since the output of P_{i^*} in the real-world experiment is derived from identically-distributed state via the same procedure (but transmitted directly to $\mathcal{Z}$ instead of $\mathcal{W}_{\mathsf{DS}}(\mathcal{F}_{\mathsf{bc}})$), it *must* be the case that the output of P_{i^*} in the real world is distributed identically to the output of $\tilde{P}_{i^*}$ (and all other honest dummy parties) in the ideal world. Thus, Lemma 4.8 can only be contradicted if there is an additional honest party P_j such that $j \neq i^*$ and the output of P_j differs from the output of P_{i^*} in the real-world experiment. We will prove that this is impossible by examining three collectively comprehensive and mutually exclusive cases:

1. $\boldsymbol{m' \neq \perp}$ **:** In this case, it must be true that $\mathcal{V}_{i^*}^{|\mathcal{P}|} = \{m'\}$ (see Step 5 of π_{DS}, and recall that by assumption P_{i^*} is not the sender). We must show that $\mathcal{V}_j^{|\mathcal{P}|} = \{m'\}$ as well. By Lemma 4.9, we know that $\mathcal{V}_j^{|\mathcal{P}|} \subseteq \mathcal{V}_{i^*}^{|\mathcal{P}|}$, therefore $|\mathcal{V}_j^{|\mathcal{P}|}| \leq 1$. The latter fact allows us to apply Lemma 4.9 again and conclude that $\mathcal{V}_{i^*}^{|\mathcal{P}|} \subseteq \mathcal{V}_j^{|\mathcal{P}|}$, which implies in turn that $\mathcal{V}_j^{|\mathcal{P}|} = \mathcal{V}_{i^*}^{|\mathcal{P}|} = \{m'\}$, and per the instructions in Step 5 of π_{DS}, the output of P_j must be m' in the real-world experiment.

2. $\boldsymbol{m' = \perp}$ **and** $\boldsymbol{\mathcal{V}_{i^*}^{|\mathcal{P}|} = \emptyset}$ **:** By Lemma 4.9, $\mathcal{V}_j^{|\mathcal{P}|} \subseteq \mathcal{V}_{i^*}^{|\mathcal{P}|}$, which implies that $\mathcal{V}_j^{|\mathcal{P}|} = \emptyset$, which implies (via the instructions in Step 5 of π_{DS}) that the output of P_j must be $\perp$ in the real-world experiment.

3. $\boldsymbol{m' = \perp}$ **and** $\boldsymbol{|\mathcal{V}_{i^*}^{|\mathcal{P}|}| > 1}$ **:** By Lemma 4.9, either $|\mathcal{V}_j^{|\mathcal{P}|}| > 1$ or $\mathcal{V}_{i^*}^{|\mathcal{P}|} \subseteq \mathcal{V}_j^{|\mathcal{P}|}$; in the latter case, $|\mathcal{V}_{i^*}^{|\mathcal{P}|}| > 1$ implies that $|\mathcal{V}_j^{|\mathcal{P}|}| > 1$ as well. Thus the instructions in Step 5 of π_{DS} once again imply that the output of P_j must be $\perp$ in the real-world experiment.

The conjunction of the above cases yields the lemma. □

A Note on Methods for Proving Adaptive Security. Canetti et al. [CDD$^+$01] proved that for perfectly secure protocols, static security implies adaptive security, albeit with inefficient simulation (see also [DN14, ACS22]). One might wonder whether it would have been sufficient to prove static security for the Dolev-Strong protocol, and then use this result to obtain adaptive security for free. However, the transformation of Canetti et al. requires the protocol to admit a *committal round*, i.e., a specific round before the end of the protocol in which it is guaranteed that the simulator can commit to its inputs by sending them to the ideal functionality. Although Theorem 4.5 proves that π_{DS} perfectly realizes $\mathcal{F}_{\mathsf{bc}}$, we emphasize that the protocol does not admit a committal round and therefore the transformation from [CDD$^+$01] cannot be applied. Indeed, a corrupted sender may send a conflicting signed message at any round, and consequently the simulator can only commit to its input at the end of the protocol.

4.5 Attacks on Dolev-Strong Broadcast under Prior Signature Functionalities

As we mentioned in Section 1, prior signature formulations and functionalities are problematic when one attempts to construct higher-level protocols that use them. Specifically, first and second generation functionalities give the adversary a means to block honest parties from generating signatures on certain messages. In this section, we illustrate what can go wrong when one tries to prove π_{DS} using Canetti's first [Can04] or second [Can05] generation signature functionality: the attacks are not limited to simple denial-of-service, but actually negate the *agreement* property for the outputs of honest participants. We stress that the attacks we construct here do *not* correspond to attacks on the Dolev-Strong broadcast protocol when true EUF-CMA signatures are employed. Rather, they are modeling artifacts that are specific to the use of these signature functionalities.

Using a First-Generation Signature Functionality. Consider the first-generation signature scheme of Canetti [Can04], which we reproduce in Appendix A.1 as $\mathcal{F}_{\mathsf{sig\text{-}1st}}$), and suppose that we attempted to prove the security of π_{DS} in the $(\mathcal{F}_{\mathsf{sig\text{-}1st}}, \mathcal{F}_{\mathsf{pki}})$-hybrid model. When an honest party requests a signature from $\mathcal{F}_{\mathsf{sig\text{-}1st}}$, activation is passed to the adversary, who is expected to return a signature string for $\mathcal{F}_{\mathsf{sig\text{-}1st}}$ to output. This implies that the adversary learns the message m^* that an honest party wishes to sign, before the signing process is actually complete. Instead of providing a signature string and returning activation to the honest party, the adversary can block the honest party from *ever* receiving a signature on m^* using the `verify` interface of $\mathcal{F}_{\mathsf{sig\text{-}1st}}$. Specifically, when the adversary receives $(\texttt{sign}, \mathsf{sid}, m^*)$ from $\mathcal{F}_{\mathsf{sig\text{-}1st}}$ for the first time, it activates the environment, and the environment causes an arbitrary party (honest or corrupt) to send $(\texttt{verify}, \mathsf{sid}, m^*, \sigma^*, v)$ to $\mathcal{F}_{\mathsf{sig\text{-}1st}}$, where v is the signer's public key and σ^* is an arbitrary value. This message is forwarded to the adversary by $\mathcal{F}_{\mathsf{sig\text{-}1st}}$, and the adversary replies with $(\texttt{verified}, \mathsf{sid}, m^*, 0)$, which causes $\mathcal{F}_{\mathsf{sig\text{-}1st}}$ to store $(m^*, \sigma^*, v, 0)$ in memory. Now the adversary (when it regains activation) replies to the original `sign` query of $\mathcal{F}_{\mathsf{sig\text{-}1st}}$ with $(\texttt{signature}, \mathsf{sid}, m^*, \sigma^*)$, which causes $\mathcal{F}_{\mathsf{sig\text{-}1st}}$ to output an error message to the honest signer.

In the context of π_{DS}, the adversary can use the above strategy to force two honest parties P_i and P_j into outputting different values, violating agreement. In round one, a corrupt sender can sign and send two different values to P_i and P_j. In round two, when P_i and P_j attempt to sign the values they received, so that they can inform the other parties of the messages they received from the sender, the adversary uses the strategy above to prevent P_i from ever receiving a verifying signature. In round three, P_i receives a valid signature from P_j, and learns that the sender transmitted two differing messages, but because P_i cannot retrieve a signature itself, P_j remains aware of only one of the sender's messages. Therefore, at the end of the protocol, P_i outputs $\perp$, while P_j outputs whatever it originally received from the sender.

Using a Second-Generation Signature Functionality. Next, consider the the second-generation signature scheme of Canetti [Can05], which we reproduce in Appendix A.2 as $\mathcal{F}_{\mathsf{sig\text{-}2nd}}$, and suppose that we attempted to prove the security of π_{DS} in the $(\mathcal{F}_{\mathsf{sig\text{-}2nd}}, \mathcal{F}_{\mathsf{pki}})$-hybrid model. In $\mathcal{F}_{\mathsf{sig\text{-}2nd}}$, activation is *not* passed to the adversary when the `sign` interface is invoked. Instead, the functionality uses signing and verification algorithms that were supplied by the adversary during the key-generation phase. In order to guarantee completeness, the functionality verifies all signatures it generates before returning them to the signer, and if a candidate signature does not pass verification, then $\mathcal{F}_{\mathsf{sig\text{-}2nd}}$ returns an error instead. The adversary can use this completeness check to prevent an honest party from receiving signatures. Consider an adversary that supplies signing and verification algorithms that are *not* EUF-CMA secure. In particular, consider an adversarial verification algorithm that always rejects signatures on certain messages (or all messages) that will be queried by some honest party P_i.[46] This prevents P_i from ever acquiring signatures on those messages, which means that the adversary can use the same strategy as it did with $\mathcal{F}_{\mathsf{sig\text{-}1st}}$ to induce differing outputs between two honest parties.

Why Consistent Verification is Necessary for the Dolev-Strong Protocol. Suppose we attempted to prove the security of Dolev-Strong using a signature scheme that lacks consistency; i.e., suppose that an adversary could maliciously pick a verification key, a message, and a signature such that verification passes with some constant probability $0 < \alpha < 1$ (see [GKZ10] for an example). It is easy to see that the Dolev-Strong protocol fails to achieve agreement under this signature scheme. Consider an adversary that corrupts the sender and all other participants, apart from two: P_i and P_j. For the sender, it picks a key, a message m, and a signature σ such that the signature verifies with probability $0 < \alpha < 1$ for some constant α. The other participants' public keys are chosen honestly. The adversary does not send any messages to P_i or P_j until round $|\mathcal{P}| - 1$ of the protocol. In round round $|\mathcal{P}| - 1$, it sends a message m to P_i, together with a valid signature chain that starts with the sender's signature σ and includes signatures from all of the other adversarial participants as well. With probability α, P_i will verify all the signatures, append her own signature, and send the result to P_j in round $|\mathcal{P}|$. In that case, P_i outputs m at the end of the protocol, but P_j *fails* to verify σ with probability $1 - \alpha$ and outputs $\perp$. Thus, with (constant) probability $\alpha(1 - \alpha)$, the protocol does not achieve agreement.

5 Generalizing to the Threshold Setting

In this section, we show that our approach can be generalized to the threshold setting. Specifically, we define a simple threshold variation of our functionality $\mathcal{F}_{\mathsf{sig}}$, and then prove that it is realized almost trivially by another functionality that internally computes the Gen and Sign algorithms of a correct, consistent, linear-time EUF-CMA signature in a threshold fashion.[47] Since we are now dealing with "non-local" functionalities that model interactive protocols rather than noninteractive algorithms (as $\mathcal{F}_{\mathsf{sig}}$ does), we must decide what kind of protocol behavior our functionality ought to enforce. Specifically, the literature of multiparty computation describes several potential guarantees in the presence of adversarial misbehavior. The most common, from weakest to strongest, are:

1. Selective Abort, which permits the adversary to cause honest parties to abort individually. That is, the honest parties do not necessarily agree whether an abort has occurred.
2. Unanimous Abort, which insists that the honest parties agree whether an abort has occurred.
3. Identifiable Abort is like unanimous abort, except that in order to cause an abort, the adversary must select one of its corrupted parties, and the identity of this party is revealed to all honest parties. That is, the honest parties agree on the identity of a corrupt party who *caused* the abort.
4. Guaranteed Output Delivery,[48] which insists that the adversary cannot cause the functionality

[46] Rejection could in principle be based upon an arbitrary predicate on the message; notice that in the Dolev-Strong protocol, honest parties signing highly structured messages that always include (for example) their own identity.

[47] According to the taxonomy that we introduced in Section 1.2, the latter functionality is a *third*-generation threshold signing functionality, and the former one, i.e., the final one, is *fourth*-generation.

[48] Otherwise known as "full security."

to abort under any circumstances. Note that guaranteed output is implied in the traditional definitions of broadcast and Byzantine agreement. Our $\mathcal{F}_{\mathsf{bc}}$ exhibits guaranteed output. Unlike the other security guarantees, guaranteed output *cannot* be achieved for certain tasks if more than half of the participating parties are corrupt [Cle86]. Since guaranteed output implies guaranteed termination, it is not captured by the plain UC model, and an additional framework such as that of Katz et al. [KMTZ13] is required.

We note that although threshold variants of ideal signatures have been proposed in the past, they inhabit a dichotomy. First- and second-generation threshold signature functionalities [CGG$^+$20, Mak22] cannot truly achieve the notions of identifiable abort or guaranteed output, as they are commonly understood, because the errors present in first- and second-generation signature functionality formulations allow the adversary to halt the functionality without output and without identifying a cheater. On the other hand, Groth and Shoup [GS22] propose a third-generation threshold signing functionality that *does* exhibit guaranteed output delivery, but it comes with the same drawbacks as all other third-generation ideal signatures, which we discussed in Section 1.2. We focus on the setting where guaranteed output is required. Our threshold signature functionality $\mathcal{F}_{\mathsf{tsig}}$ features a random mode, like our $\mathcal{F}_{\mathsf{sig}}$, and we prove that it can be realized via a third-generation threshold signing functionality with guaranteed output, such as Groth and Shoup's [GS22]. More specifically, we prove:

Theorem 5.1 (Informal Version of Theorem 5.5)**.** *If there is a functionality that simply computes the* Gen *and* Sign *algorithms of a signature scheme with guaranteed output when it is invoked by any t out of n designated signers, then the trivial protocol that invokes that functionality and also allows any party to verify a signature by running the* Verify *algorithm locally UC-realizes* $\mathcal{F}_{\mathsf{tsig}}$ *in the presence of a malicious adversary that adaptively corrupts any set of parties.*

We stress, however, that our proof should be interpreted as a template: it can be adjusted to show that weaker third-generation functionalities (e.g., functionalities attaining only identifiable abort, unanimous abort, or selective abort) imply correspondingly weaker fourth-generation ones.

5.1 A Fourth-Generation Threshold Signature Functionality

We begin by giving our *fourth*-generation threshold signing functionality $\mathcal{F}_{\mathsf{tsig}}$, which is analogous to the $\mathcal{F}_{\mathsf{sig}}$ we introduced in Section 3. We note that since we wish to achieve guaranteed output delivery (which implies guaranteed termination), we once again use the framework of Katz et al. [KMTZ13] and the round-extending wrapper scheme of Cohen et al. [CCGZ16].

Functionality 5.2. $\mathcal{F}_{\mathsf{tsig}}(t)$ (An Unstoppable *Threshold* Signature Functionality)

This functionality interacts with an ideal adversary $\mathcal{S}$ and with n designated signer parties $P_1, \ldots, P_n$, plus a number of verifier parties that is not a-priori known. For simplicity of description, we assume this functionality has *per-session* memory. That is, all stored and recalled values are associated with the particular session ID sid of the query that generated them. This functionality emulates the functionality $\mathcal{F}_{\mathsf{sig}}$ internally, and in essence *distributes* its interfaces to the appropriate sets of participants. All messages received directly from the adversary $\mathcal{S}$[a] are forwarded to the emulated $\mathcal{F}_{\mathsf{sig}}$ instance, *except* for `corrupt` messages,[b] and all messages that $\mathcal{F}_{\mathsf{sig}}$ writes on its back door tape are forwarded to $\mathcal{S}$. Note that this functionality sends additional messages to $\mathcal{S}$, beyond those produced by $\mathcal{F}_{\mathsf{sig}}$. Apart from the interactions conducted via the back door tape, all interactions with $\mathcal{F}_{\mathsf{sig}}$ are conducted via emulated parties; one party is emulated to $\mathcal{F}_{\mathsf{sig}}$ for each party that sends input to $\mathcal{F}_{\mathsf{tsig}}$, and in addition, $\mathcal{F}_{\mathsf{tsig}}$ emulates a distinct, special party P_{sid}.

This functionality achieves synchrony through the framework of Katz et al. [KMTZ13], i.e., it only proceeds from one round to the next when all honest parties have indicated that it should do so. An honest party can be adaptively corrupted at any point. In every round, the functionality ignores all messages other than the ones we explicitly specify that it can receive.

Initialization.

1. Ignore any message from any party P that contains some session ID sid until *after* party P sends $(\texttt{init}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{tsig}}$.
2. Upon receiving $(\texttt{init}, \mathsf{sid})$ for the *first time* for some particular sid, send $(\texttt{init}, \mathsf{sid})$ to the internally-emulated $\mathcal{F}_{\mathsf{sig}}$ instance on behalf of P_{sid}. Additionally, forward all $\texttt{init}$ commands to $\mathcal{F}_{\mathsf{sig}}$ on behalf of the party that transmitted them.

Key Generation.

3. **Input Round:** Upon receiving $(\texttt{keygen}, \mathsf{sid})$ from some signer P_i, send $(\texttt{keygen}, \mathsf{sid}, P_i)$ to $\mathcal{S}$. If P_i is the t^{th} signing party to send a $\texttt{keygen}$ message for this sid, then forward it to the internally emulated $\mathcal{F}_{\mathsf{sig}}$ instance on behalf of P_{sid}, receive $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk})$ in response, and store the latter message in memory.
4. **Output Round:** Upon receiving $(\texttt{fetch-output}, \mathsf{sid})$ from some signer P_i, if $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk})$ is stored in memory, then send $(\mathsf{output}, \mathsf{sid}, \mathsf{pk})$ to P_i.

Signing.

5. **Input Round:** Upon receiving $(\texttt{sign}, \mathsf{sid}, \mathsf{ssid}, \mathsf{pk}, m)$ from some signing party P_i such that P_i has not previously used this ssid, send $(\texttt{sign}, \mathsf{sid}, \mathsf{ssid}, P_i)$ to $\mathcal{S}$. If P_i is the t^{th} signer to send a $\texttt{sign}$ message for this sid and ssid, and all $t-1$ previous messages have the same pk and m, then send $(\texttt{sign}, \mathsf{sid}, \mathsf{pk}, m)$ to the internally-emulated $\mathcal{F}_{\mathsf{sig}}$ instance on behalf of P_{sid}, receive $(\texttt{signature}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ in response, and store $(\texttt{signature}, \mathsf{sid}, \mathsf{ssid}, \mathsf{pk}, m, \sigma)$ in memory.
6. **Output Round:** Upon receiving $(\texttt{fetch-output}, \mathsf{sid}, \mathsf{ssid})$ from some signer P_i, if $(\texttt{signature}, \mathsf{sid}, \mathsf{ssid}, \mathsf{pk}, m, \sigma)$ is stored in memory, then send $(\mathsf{output}, \mathsf{sid}, \mathsf{ssid}, \sigma)$ to P_i.

Verification.

7. Upon receiving $(\texttt{verify}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$ from any party P,[c] forward this message to the internally emulated $\mathcal{F}_{\mathsf{sig}}$ on behalf of P, and forward the response to P.

Corruption.

8. Upon receiving $(\texttt{corrupt}, \mathsf{sid}, P_i)$ from $\mathcal{S}$, if this is the t^{th} such message to be sent with this sid, and all t messages identify distinct parties from the set of signers $P_1, \ldots P_n$, then send $(\texttt{corrupt}, \mathsf{sid}, P_{\mathsf{sid}})$ to $\mathcal{F}_{\mathsf{sig}}$ on behalf of $\mathcal{S}$.

[a]i.e., messages not sent by $\mathcal{S}$ on behalf of some corrupt party.
[b]`corrupt` messages are handled explicitly in step 8.
[c]P is not necessarily one of the designated signers $P_1, \dots P_n$

5.2 Realizing $\mathcal{F}_{\mathsf{tsig}}$ via a Third-Generation Signing Functionality

Now we give a *third*-generation threshold signing functionality $\mathcal{F}_{\mathsf{tGenSign}}$, which we will use to realize the previous functionality. Briefly, it allows any size-t subset of the signers to compute the output $\mathsf{Sign}(\mathsf{sk}, m; r_\sigma)$ of the signing algorithm of a EUF-CMA signature scheme. We note that $\mathcal{F}_{\mathsf{tGenSign}}$ is not quite the same as the functionality given by Groth and Shoup [GS22], even though we propose to use their protocol to realize our functionality. We discuss how to resolve this mismatch in Section 5.4.

Functionality 5.3. $\mathcal{F}_{\mathsf{tGenSign}}(t)$ (Threshold Signature Calculation)

This functionality interacts with n designated signer parties $P_1, \dots, P_n$. For simplicity of description, we assume this functionality has *per-session* memory. That is, all stored and recalled values are associated with the particular session ID sid of the query that generated them.

This functionality achieves synchrony through the framework of Katz et al. [KMTZ13], i.e., it only proceeds from one round to the next when all honest parties have indicated that it should do so. An honest party can be adaptively corrupted at any point. In every round, the functionality ignores all messages other than the ones we explicitly specify that it can receive.

Key Generation.

1. **Input Round:** Upon receiving $(\texttt{keygen}, \mathsf{sid})$ from some signer P_i, send $(\texttt{keygen}, \mathsf{sid}, P_i)$ to $\mathcal{S}$. If P_i is the t^{th} signing party to send a $\texttt{keygen}$ message for this sid, then sample a uniformly random bit-string r_k of appropriate length, run $(\mathsf{sk}, \mathsf{pk}) := \mathsf{Gen}(1^\kappa; r_k)$, and store $(\texttt{key}, \mathsf{sid}, \mathsf{pk}, \mathsf{sk}, r_k)$ in memory.
2. **Output Round:** Upon receiving $(\texttt{fetch-output}, \mathsf{sid})$ from some signer P_i, if $(\texttt{key}, \mathsf{sid}, \mathsf{pk}, *, *)$ is stored in memory, then send $(\mathsf{output}, \mathsf{sid}, \mathsf{pk})$ to P_i.

Signing.

3. **Input Round:** Upon receiving $(\texttt{sign}, \mathsf{sid}, \mathsf{ssid}, \mathsf{pk}, m)$ from some signing party P_i such that P_i has not previously used this ssid, send $(\texttt{sign}, \mathsf{sid}, \mathsf{ssid}, P_i)$ to $\mathcal{S}$. If P_i is the t^{th} signer to send a $\texttt{sign}$ message for this sid and ssid, and all $t-1$ previous messages have the same pk and m, and the record $(\texttt{key}, \mathsf{sid}, \mathsf{pk}, \mathsf{sk}, *)$ exists in memory, then sample a uniformly random bit-string r_σ of appropriate length, compute $\sigma := \mathsf{Sign}(\mathsf{sk}, m; r_\sigma)$, and store $(\texttt{sig}, \mathsf{sid}, \mathsf{ssid}, \mathsf{pk}, m, \sigma, r_\sigma)$ in memory.
4. **Output Round:** Upon receiving $(\texttt{fetch-output}, \mathsf{sid}, \mathsf{ssid})$ from some signer P_i, if $(\texttt{sig}, \mathsf{sid}, \mathsf{ssid}, \mathsf{pk}, m, \sigma, *)$ is stored in memory, then send $(\mathsf{output}, \mathsf{sid}, \mathsf{ssid}, \sigma)$ to P_i.

Corruption.

5. Upon receiving $(\texttt{corrupt}, \mathsf{sid}, P_i)$ from $\mathcal{S}$, if this is the t^{th} such message to be sent with this sid, and all t messages identify distinct parties from the set of signers $P_1, \ldots P_n$, then search the memory for all records of the form $(\texttt{key}, \mathsf{sid}, \mathsf{pk}, \mathsf{sk}, r_k)$, and for each such record compute the set $\mathcal{C}_{\mathsf{pk}}$ of all $(m, \mathsf{ssid}, \sigma, r_\sigma)$ such that there exists a record of the form $(\texttt{sig}, \mathsf{sid}, \mathsf{ssid}, \mathsf{pk}, m, \sigma, r_\sigma)$ in memory. Return $(\texttt{corrupt}, \mathsf{sid}, \mathcal{C})$ to $\mathcal{S}$, where $\mathcal{C}$ is a set containing $(\mathsf{pk}, \mathsf{sk}, r_k, \mathcal{C}_{\mathsf{pk}})$ for every $(\texttt{key}, \mathsf{sid}, \mathsf{pk}, \mathsf{sk}, r_k)$ that was found.

Because we are using one functionality ($\mathcal{F}_{\mathsf{tGenSign}}$) to realize another one ($\mathcal{F}_{\mathsf{tsig}}$) without any additional protocol code,[49] it is essentially the case that a single protocol realizes them both. We wish the functionalities to be independent of the round structure of such a protocol, and we wish the proof to be independent of the concrete protocol details entirely. Thus we give a round-extending wrapper $\mathcal{W}_{\mathsf{tGenSign}}$ that is compatible with *both* $\mathcal{F}_{\mathsf{tsig}}$ *and* $\mathcal{F}_{\mathsf{tGenSign}}$. Unlike our previous wrapper $\mathcal{W}_{\mathsf{DS}}$ (see Section 4.2), which had a hard-coded round count, $\mathcal{W}_{\mathsf{tGenSign}}$ is parameterized by two distributions $\mathcal{D}_{\mathsf{tGen}}$ and $\mathcal{D}_{\mathsf{tSign}}$ that model the round counts of the $\texttt{keygen}$ and $\texttt{sign}$ operations (respectively) of whatever protocol realizes $\mathcal{W}_{\mathsf{tGenSign}}^{\mathcal{D}_{\mathsf{tGen}}, \mathcal{D}_{\mathsf{tSign}}}(\mathcal{F}_{\mathsf{tGenSign}})$. We note that this wrapper passes commands through to the functionality it wraps by default; this is what allows it to host two syntactically-mismatched threshold signing functionalities.

Functionality 5.4. $\mathcal{W}_{\mathsf{tGenSign}}^{\mathcal{D}_{\mathsf{tGen}}, \mathcal{D}_{\mathsf{tSign}}}(\mathcal{F})$ (The Strict Wrapper for Threshold Signing [CCGZ16])

The wrapper functionality $\mathcal{W}_{\mathsf{tGenSign}}(\mathcal{F})$ interacts with an ideal adversary $\mathcal{S}$ and designated signing parties $P_1, \ldots, P_n$, plus a number of additional non-signing parties (all of whom are denoted P) that is not a-prior bounded. It has oracle access to two distributions $\mathcal{D}_{\mathsf{tGen}}$ and $\mathcal{D}_{\mathsf{tSign}}$. It runs a copy of some parametrically-defined functionality $\mathcal{F}$ internally. All messages from external entities that are not explicitly handled below are immediately forwarded to $\mathcal{F}$ on behalf of the sender, and all messages from $\mathcal{F}$ that are not explicitly handled are immediately forwarded to the corresponding external entity.

Key Generation.

1. Upon receiving the *first* message of the form $(\texttt{keygen}, \mathsf{sid})$ for a fresh sid set the output round $\rho_{\mathsf{Gen}} \leftarrow \mathcal{D}_{\mathsf{tGen}}$, and send $(\mathsf{output\text{-}round}, \mathsf{sid}, \rho_{\mathsf{Gen}})$ to the adversary. Continue processing this message using the rules enumerated below.
2. At all times, forward $(\texttt{keygen}, \mathsf{sid})$ messages from corrupted parties to $\mathcal{F}$.
3. In round $\rho = 1$: Forward $(\texttt{keygen}, \mathsf{sid})$ messages from all honest signing parties to $\mathcal{F}$.
4. In rounds $\rho > 1$: Upon receiving a message $(\texttt{fetch-output}, \mathsf{sid})$ from some signing party P_i:
   - If $\rho = \rho_{\mathsf{Gen}}$, forward the message to $\mathcal{F}$, and the response $(\mathsf{output}, \mathsf{sid}, *)$ to P_i.
   - Otherwise, send $(\texttt{fetch-output}, \mathsf{sid}, P_i)$ to the adversary.

Signing.

[49] We need a dummy protocol to handle verification, as we will discuss in Section 5.3, but key generation and signing commands are simply passed through.

5. Upon receiving the *first* message of the form $(\texttt{sign}, \mathsf{sid}, \mathsf{ssid}, *, *)$ for a fresh ssid, set the output round $\rho_{\mathsf{ssid}} := \mathcal{D}_{\mathsf{tSign}}$,[a] and send $(\mathsf{output\text{-}round}, \mathsf{sid}, \mathsf{ssid}, \rho_{\mathsf{ssid}})$ to the adversary. Continue processing this message using the rules enumerated below.
6. At all times, forward $(\texttt{sign}, \mathsf{sid}, \mathsf{ssid}, *, *)$ messages from corrupted parties to $\mathcal{F}$.
7. In round $\rho = 1$: Forward $(\texttt{sign}, \mathsf{sid}, \mathsf{ssid}, *, *)$ messages from all honest signing parties to $\mathcal{F}$.
8. In rounds $\rho > 1$: Upon receiving a message $(\texttt{fetch-output}, \mathsf{sid}, \mathsf{ssid})$ from some signing party P_i:
   - If $\rho = \rho_{\mathsf{ssid}}$, forward the message to $\mathcal{F}$, and the response $(\mathsf{output}, \mathsf{sid}, \mathsf{ssid}, *)$ to P_i.
   - Otherwise, send $(\texttt{fetch-output}, \mathsf{sid}, \mathsf{ssid}, P_i)$ to the adversary.

[a]Note that there is one variable ρ_{ssid} per ssid value.

5.3 Proof of Security

A Real-World Dummy Protocol. As in Section 3.2, we specify a simple *dummy protocol* $\pi_{\mathsf{t\Sigma}}(t)$ that converts a signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *and* the functionality $\mathcal{W}^{\mathcal{D}_{\mathsf{tGen}}, \mathcal{D}_{\mathsf{tSign}}}_{\mathsf{tGenSign}}(\mathcal{F}_{\mathsf{tGenSign}}(t))$ that runs Σ internally into a real-world UC experiment that is a syntactic match for $\mathcal{W}^{\mathcal{D}_{\mathsf{tGen}}, \mathcal{D}_{\mathsf{tSign}}}_{\mathsf{tGenSign}}(\mathcal{F}_{\mathsf{tsig}}(t))$. Each party ignores all instructions from the environment in the session sid until after it receives $(\texttt{init}, \mathsf{sid})$. Thereafter, the parties forward $\texttt{keygen}$ and $\texttt{sign}$ commands to $\mathcal{W}^{\mathcal{D}_{\mathsf{tGen}}, \mathcal{D}_{\mathsf{tSign}}}_{\mathsf{tGenSign}}(\mathcal{F}_{\mathsf{tGenSign}}(t))$, and output the functionality's responses to the environment. If any party receives $(\texttt{verify}, \mathsf{sid}, \mathsf{pk}, m, \sigma)$, it sets $b \leftarrow \mathsf{Verify}(1^\kappa, \mathsf{pk}, m, \sigma)$ and returns $(\texttt{verified}, \mathsf{sid}, \mathsf{pk}, m, \sigma, b)$ to the environment.

Theorem 5.5. *If $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ is a correct (Definition 2.2), consistent (Definition 2.4), EUF-CMA secure (Definition 2.3), linear-time signature scheme (Definition 2.5), then $\pi_{\mathsf{t\Sigma}}(t)$ UC-realizes $\mathcal{W}^{\mathcal{D}_{\mathsf{tGen}}, \mathcal{D}_{\mathsf{tSign}}}_{\mathsf{tGenSign}}(\mathcal{F}_{\mathsf{tsig}}(t))$ in the $\mathcal{W}^{\mathcal{D}_{\mathsf{tGen}}, \mathcal{D}_{\mathsf{tSign}}}_{\mathsf{tGenSign}}(\mathcal{F}_{\mathsf{tGenSign}}(t))$-hybrid model against a malicious adversary that adaptively corrupts any set of parties.*

Proof. Formally, we will prove that for every adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}_{\mathsf{tsig}}$ such that for any environment $\mathcal{Z}$ and any $n, t \in \mathbb{N}$ such that $n \geq t \geq 1$,

$$\left\{\textsc{Real}_{\pi_{\mathsf{t\Sigma}}(t), \mathcal{A}, \mathcal{Z}}(\kappa, z)\right\}_{\kappa \in \mathbb{N}, z \in \{0,1\}^*} \approx_{\mathrm{c}} \left\{\textsc{Ideal}_{\mathcal{W}^{\mathcal{D}_{\mathsf{tGen}}, \mathcal{D}_{\mathsf{tSign}}}_{\mathsf{tGenSign}}(\mathcal{F}_{\mathsf{tsig}}(t)), \mathcal{S}_{\mathsf{tsig}}, \mathcal{Z}}(\kappa, z)\right\}_{\kappa \in \mathbb{N}, z \in \{0,1\}^*}.$$

Our proof proceeds via a sequence of hybrid experiments, which begins with

$$\mathcal{H}_0 = \left\{\textsc{Real}_{\pi_{\mathsf{t\Sigma}}(t), \mathcal{A}, \mathcal{Z}}(\kappa, z)\right\}_{\kappa \in \mathbb{N}, z \in \{0,1\}^*}$$

and gradually modifies the behavior of the honest parties until they invoke $\mathcal{F}_{\mathsf{tsig}}$ instead.

Hybrid $\mathcal{H}_1$. This experiment is like $\mathcal{H}_0$, except that we run a secondary *sub*experiment in parallel with the primary one, which contains an instance of the protocol π_{Σ}. Most of the remaining hybrids in this sequence will feature the same structure: in parallel with each experiment, we run a subexperiment. In $\mathcal{H}_1$, the subexperiment is *independent* of the primary experiment. In future hybrids, we will connect the two to one another, and eventually merge them. The subexperiment contains one party for each party in the primary experiment. We denote by P'_i the party in the subexperiment that corresponds to P_i in the primary experiment. Finally, the subexperiment contains an extra party P'_{sid} that has no analogue in the primary experiment. The subexperiment has no environment or adversary per se, and cannot be accessed by $\mathcal{Z}$ or $\mathcal{A}$ from the primary experiment. Consequently, the primary experiment behaves exactly as in $\mathcal{H}_0$. Thus we have $\mathcal{H}_1 = \mathcal{H}_0$.

Hybrid $\mathcal{H}_2$. In this experiment, we make several changes relative to $\mathcal{H}_1$:

1. Whenever P in the primary experiment receives an `init` instruction from the environment, it forwards that command to the *input tape* of the corresponding P' in the subexperiment, and activates P'.

2. When a party P in the primary experiment receives a `verify` instruction from the environment, rather than running the code of $\pi_{\mathsf{t}\Sigma}(t)$ as in $\mathcal{H}_1$, it instead forwards that instruction to the input tape of P' in the subexperiment. P' in turn forwards its outputs to P, and the latter writes them on its own output tape.

3. When $\mathcal{W}^{\mathcal{D}_{\mathsf{tGen}},\mathcal{D}_{\mathsf{tSign}}}_{\mathsf{tGenSign}}(\mathcal{F}_{\mathsf{tGenSign}})$ runs $(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{Gen}(1^\kappa)$ or $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$, respectively in $\mathcal{H}_1$, in $\mathcal{H}_2$ it instead writes $(\texttt{keygen}, \mathsf{sid})$ or $(\texttt{sign}, \mathsf{sid}, \mathsf{ssid}, \mathsf{pk}, m)$ on the input tape of P'_{sid} in the subexperiment. P'_{sid} returns its outputs to $\mathcal{F}_{\mathsf{tGenSign}}$, and the latter uses them as though they were produced by the algorithms.

4. When $\mathcal{W}^{\mathcal{D}_{\mathsf{tGen}},\mathcal{D}_{\mathsf{tSign}}}_{\mathsf{tGenSign}}(\mathcal{F}_{\mathsf{tGenSign}})$ receives the t^{th} $(\texttt{corrupt}, \mathsf{sid}, P_i)$ message from $\mathcal{A}$ in the primary experiment such that P_i is a signer, it does not follow its specified code, but instead writes $(\texttt{corrupt}, \mathsf{sid}, P'_{\mathsf{sid}})$ on the input tape of a *dummy adversary* $\mathcal{D}'$ in the subexperiment, and forwards the response of $\mathcal{D}'$ to $\mathcal{A}$.

Although we have made many changes, our argument is relatively straightforward. $\mathcal{W}^{\mathcal{D}_{\mathsf{tGen}},\mathcal{D}_{\mathsf{tSign}}}_{\mathsf{tGenSign}}(\mathcal{F}_{\mathsf{tGenSign}}(t))$ emulates internally exactly the instructions of a single party in the π_{Σ} protocol, and outputs the state of that party exactly when t participants are corrupted. The other changes we have made are purely syntactical in the sense that all instructions from the environment result in exactly the same code being run in $\mathcal{H}_2$ and $\mathcal{H}_1$ (the code is run by different turing machines, but the environment cannot detect this fact). It follows that $\mathcal{H}_2 = \mathcal{H}_1$.

Hybrid $\mathcal{H}_3$. In $\mathcal{H}_3$, we replace the subexperiment, which comprises an instance of the π_{Σ} protocol, with a subexperiment comprising the corresponding instance of the ideal protocol involving $\mathcal{F}_{\mathsf{sig}}$. The primary experiment in both $\mathcal{H}_3$ and $\mathcal{H}_2$ can be interpreted as an environment for the subexperiment. Therefore, by Theorem 3.2, there exists some simulator $\mathcal{S}'$ for the dummy adversary $\mathcal{D}'$ such that $\mathcal{H}_3 \approx_{\mathrm{c}} \mathcal{H}_2$.

Hybrid $\mathcal{H}_4$. In $\mathcal{H}_4$, we simplify and merge the primary experiment and the subexperiment. The dummy parties in the subexperiment are removed. Instead, each party in the primary experiment (and the functionality $\mathcal{W}^{\mathcal{D}_{\mathsf{tGen}},\mathcal{D}_{\mathsf{tSign}}}_{\mathsf{tGenSign}}(\mathcal{F}_{\mathsf{tGenSign}}(t))$) send directly to $\mathcal{F}_{\mathsf{sig}}$ whatever messages they would have sent to dummy parties. This is a purely syntactic change, and thus $\mathcal{H}_4 = \mathcal{H}_3$. We observe also that this experiment behaves identically to the ideal world experiment involving $\mathcal{F}_{\mathsf{tsig}}(t)$.[50] Thus we have

$$\mathcal{H}_4 = \left\{ \text{IDEAL}_{\mathcal{W}^{\mathcal{D}_{\mathsf{tGen}},\mathcal{D}_{\mathsf{tSign}}}_{\mathsf{tGenSign}}(\mathcal{F}_{\mathsf{tsig}}(t)),\mathcal{S}_{\mathsf{tsig}},\mathcal{Z}}(\kappa, z) \right\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}$$

[50]There remain a few unimportant syntactical distinctions.

and Theorem 5.5 follows by transitivity.[51] □

5.4 On Realizing Our Third-Generation Threshold Signature Functionality

Finally, we note that there does not exist any protocol in the literature that realizes our functionality $\mathcal{F}_{\mathsf{tGenSign}}(t)$ as written. The closest candidate protocol is that of Groth and Shoup [GS22], which would realize our functionality were it not for the fact that their protocol allows the adversary to condition the message on part of the signature, because it releases *presignatures* before the message is committed. Simply insisting that all honest parties commit[52] to the message that they wish to sign before the presignatures are released solves this mismatch and allows us to instantiate $\mathcal{F}_{\mathsf{tGenSign}}$ (and therefore $\mathcal{F}_{\mathsf{tsig}}$) for ECDSA signatures, in the presence of an adversary that corrupts up to $n/3$ signers.

If we relax our security requirement from guaranteed output to identifiable abort,[53] then the protocol Cohen et al. [CDKS24] realizes the relaxed functionality for ECDSA signatures. We stress the distinction between this result and the result of Canetti et al. [CGG+20], who realized a *second-generation* ideal threshold signing functionality via an ECDSA signing protocol with identifiable abort: the Canetti et al. functionality includes a *non-identifiable* error, which gives the adversary a mechanism to deny the signers a signature *without* revealing the identity of a corrupt party (or, indeed, corrupting a party at all). By comparison, our approach would achieve a stricter and more intuitive notion of identifiability: either the parties output a signature, or they output the identity of a corrupt party.

Acknowledgments

The work of Ran Cohen was supported in part by NSF grant no. 2055568 and by ISF grant 1834/23. The work of Jack Doerner and Eysa Lee was supported by the Brown University Data Science Institute. Anna Lysyanskaya is supported by NSF Grants 2312241, 2154170, and 2247305 as well as the Ethereum Foundation. The work of Lawrence Roy was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement number 803096 "SPEC"; the Danish Independent Research Council under Grant-ID DFF-0165-00107B "C3PO"; and the DARPA SIEVE program (contract HR001120C0085 "FROMAGER"). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding entities. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited).

References

[ACS22] Gilad Asharov, Ran Cohen, and Oren Shochat. Static vs. adaptive security in perfect MPC: A separation and the adaptive security of BGW. In *3rd Conference on Information-Theoretic Cryptography, ITC 2022*, volume 230 of *LIPIcs*, pages 15:1–15:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[BCH+20] Christian Badertscher, Ran Canetti, Julia Hesse, Björn Tackmann, and Vassilis Zikas. Universal composition with global subroutines: Capturing global setup within plain

[51] Notice that $\mathcal{S}_{\mathsf{tsig}}$ is just $\mathcal{S}'$.

[52] Not necessarily in a message-hiding way.

[53] As we mentioned at the beginning of this section, our proof can be modified to show that a variation of $\mathcal{F}_{\mathsf{tGenSign}}$ with identifiable abort implies a variation of our unstoppable $\mathcal{F}_{\mathsf{tsig}}$ with identifiable abort.

UC. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part III*, volume 12552 of *LNCS*, pages 1–30. Springer, Heidelberg, November 2020.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Heidelberg, August 2001.

[BH04] Michael Backes and Dennis Hofheinz. How to break and repair a universally composable signature functionality. In Kan Zhang and Yuliang Zheng, editors, *ISC 2004*, volume 3225 of *LNCS*, pages 61–72. Springer, Heidelberg, September 2004.

[BKL19] Erica Blum, Jonathan Katz, and Julian Loss. Synchronous consensus with optimal asynchronous fallback guarantees. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 131–150. Springer, Heidelberg, December 2019.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[Can04] Ran Canetti. Universally composable signature, certification, and authentication. In *17th IEEE Computer Security Foundations Workshop, (CSFW)*, pages 219–233. IEEE Computer Society, 2004.

[Can05] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Version of December 2005, 2005. https://eccc.weizmann.ac.il/eccc-reports/2001/TR01-016.

[Can20] Ran Canetti. Universally composable security. *Journal of the ACM*, 67(5):28:1–28:94, 2020.

[CCGZ16] Ran Cohen, Sandro Coretti, Juan A. Garay, and Vassilis Zikas. Probabilistic termination and composability of cryptographic protocols. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 240–269. Springer, Heidelberg, August 2016.

[CCGZ17] Ran Cohen, Sandro Coretti, Juan A. Garay, and Vassilis Zikas. Round-preserving parallel composition of probabilistic-termination cryptographic protocols. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *ICALP 2017*, volume 80 of *LIPIcs*, pages 37:1–37:15, July 2017.

[CDD$^+$01] Ran Canetti, Ivan Damgård, Stefan Dziembowski, Yuval Ishai, and Tal Malkin. On adaptive vs. non-adaptive security of multiparty protocols. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 262–279. Springer, Heidelberg, May 2001.

[CDKS24] Ran Cohen, Jack Doerner, Yashvanth Kondi, and Abhi Shelat. Secure multiparty computation with identifiable abort via vindicating release. LNCS, pages 36–73. Springer, Heidelberg, August 2024.

[CGG$^+$20] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1769–1787. ACM Press, November 2020.

[CGZ23] Ran Cohen, Juan A. Garay, and Vassilis Zikas. Completeness theorems for adaptively secure broadcast. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 3–38. Springer, Heidelberg, August 2023.

[CKKR19] Jan Camenisch, Stephan Krenn, Ralf Küsters, and Daniel Rausch. iUC: Flexible universal composability made simple. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 191–221. Springer, Heidelberg, December 2019.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *18th ACM STOC*, pages 364–369. ACM Press, May 1986.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002.

[CR03] Ran Canetti and Tal Rabin. Universal composition with joint state. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 265–281. Springer, Heidelberg, August 2003.

[CSV16] Ran Canetti, Daniel Shahaf, and Margarita Vald. Universally composable authentication and key-exchange with global PKI. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 265–296. Springer, Heidelberg, March 2016.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DKL$^+$23] Jack Doerner, Yashvanth Kondi, Eysa Lee, abhi shelat, and LaKyah Tyner. Threshold BBS+ signatures for distributed anonymous credential issuance. In *2023 IEEE Symposium on Security and Privacy*, pages 773–789. IEEE Computer Society Press, May 2023.

[DKLs18] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *2018 IEEE Symposium on Security and Privacy*, pages 980–997. IEEE Computer Society Press, May 2018.

[DKLS19] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *2019 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2019.

[DKLS24] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ECDSA in three rounds. In *2024 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2024.

[DN14] Ivan Damgård and Jesper Buus Nielsen. Adaptive versus static security in the UC model. In Sherman S. M. Chow, Joseph K. Liu, Lucas C. K. Hui, and Siu-Ming Yiu, editors, *ProvSec 2014*, volume 8782 of *LNCS*, pages 10–28. Springer, Heidelberg, October 2014.

[DR82] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. In Robert L. Probert, Michael J. Fischer, and Nicola Santoro, editors, *1st ACM PODC*, pages 132–140. ACM, August 1982.

[DS83] Danny Dolev and H. Raymond Strong. Authenticated algorithms for Byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[DY81] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols (extended abstract). In *22nd FOCS*, pages 350–357. IEEE Computer Society Press, October 1981.

[ELP24] Fatima Elsheimy, Julian Loss, and Charalampos Papamanthou. Early stopping byzantine agreement in $(1+\epsilon) \cdot f$ rounds. Cryptology ePrint Archive, Paper 2024/822, 2024. https://eprint.iacr.org/2024/822.

[FLL21] Matthias Fitzi, Chen-Da Liu-Zhang, and Julian Loss. A new way to achieve round-efficient byzantine agreement. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *40th ACM PODC*, pages 355–362. ACM, July 2021.

[FLM86] Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1(1):26–39, 1986.

[FN09] Matthias Fitzi and Jesper Buus Nielsen. On the number of synchronous rounds sufficient for authenticated byzantine agreement. In *Distributed Computing, 23rd International Symposium, DISC 2009, Elche, Spain, September 23-25, 2009. Proceedings*, volume 5805 of *Lecture Notes in Computer Science*, pages 449–463. Springer, 2009.

[GGL22] Diana Ghinea, Vipul Goyal, and Chen-Da Liu-Zhang. Round-optimal byzantine agreement. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 96–119. Springer, Heidelberg, May / June 2022.

[GKKO07] Juan A. Garay, Jonathan Katz, Chiu-Yuen Koo, and Rafail Ostrovsky. Round complexity of authenticated broadcast with a dishonest majority. In *48th FOCS*, pages 658–668. IEEE Computer Society Press, October 2007.

[GKKZ11] Juan A. Garay, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. Adaptively secure broadcast, revisited. In Cyril Gavoille and Pierre Fraigniaud, editors, *30th ACM PODC*, pages 179–186. ACM, June 2011.

[GKZ10] Juan A. Garay, Aggelos Kiayias, and Hong-Sheng Zhou. A framework for the sound specification of cryptographic tasks. In Andrew Myers and Michael Backes, editors, *CSF 2010 Computer Security Foundations Symposium*, pages 277–289. IEEE Computer Society Press, 2010.

[GLW22] Diana Ghinea, Chen-Da Liu-Zhang, and Roger Wattenhofer. Optimal synchronous approximate agreement with asynchronous fallback. In Alessia Milani and Philipp Woelfel, editors, *41st ACM PODC*, pages 70–80. ACM, July 2022.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.

[GS22] Jens Groth and Victor Shoup. Design and analysis of a distributed ECDSA signing service. *IACR Cryptol. ePrint Arch.*, 2022. https://eprint.iacr.org/2022/506.

[HZ10] Martin Hirt and Vassilis Zikas. Adaptively secure broadcast. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 466–485. Springer, Heidelberg, May / June 2010.

[KK06] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 445–462. Springer, Heidelberg, August 2006.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.

[KMTZ13] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 477–498. Springer, Heidelberg, March 2013.

[KT08] Ralf Küsters and Max Tuengerthal. Joint state theorems for public-key encryption and digital signature functionalities with local computation. In Andrei Sabelfeld, editor, *CSF 2008 Computer Security Foundations Symposium*, pages 270–284. IEEE Computer Society Press, 2008.

[Lin17] Yehuda Lindell. Fast secure two-party ECDSA signing. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 613–644. Springer, Heidelberg, August 2017.

[Lin19] Yehuda Lindell. https://crypto.stackexchange.com/questions/71292/induction-is-problematic-in-computational-cryptography-why, 2019.

[Lin22] Yehuda Lindell. Simple three-round multiparty schnorr signing with full simulatability. Cryptology ePrint Archive, Paper 2022/374, 2022. https://eprint.iacr.org/2022/374.

[LL22] Christoph Lenzen and Julian Loss. Optimal clock synchronization with signatures. In Alessia Milani and Philipp Woelfel, editors, *41st ACM PODC*, pages 440–449. ACM, July 2022.

[LLR02] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. On the composition of authenticated byzantine agreement. In *34th ACM STOC*, pages 514–523. ACM Press, May 2002.

[LN24] Julian Loss and Jesper Buus Nielsen. Early stopping for any number of corruptions. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic*

Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part III, volume 14653 of *Lecture Notes in Computer Science*, pages 457–488. Springer, 2024.

[LSP82] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The Byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[Mak22] Nikolaos Makriyannis. On the classic protocol for MPC schnorr signatures. *IACR Cryptol. ePrint Arch.*, 2022. https://eprint.iacr.org/2022/1332.

[Pat05] Akshay Patil. *On symbolic analysis of cryptographic protocols.* PhD thesis, Massachusetts Institute of Technology, 2005.

[PSL80] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, February 1978.

[TLP22] Georgios Tsimos, Julian Loss, and Charalampos Papamanthou. Gossiping for communication-efficient broadcast. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part III*, volume 13509 of *LNCS*, pages 439–469. Springer, Heidelberg, August 2022.

[WXSD20] Jun Wan, Hanshen Xiao, Elaine Shi, and Srinivas Devadas. Expected constant round byzantine broadcast under dishonest majority. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 381–411. Springer, Heidelberg, November 2020.

A Prior Signature Functionalities

For ease of reference, we have reproduced the signature functionalities of [Can04] and the 2005 version [Can05] of the original UC paper [Can01]. While these are not the only functionalities in the literature, these two are generally representative of two flavors of signature functionalities, which we refer to as "first generation" and "second generation" in Section 1.2.

A.1 The First Generation Signature Functionality of Canetti [Can04]

Functionality A.1. $\mathcal{F}_{\textsf{sig-1st}}$ (Example First-Generation Signature Functionality)

Key Generation. Upon receiving $(\texttt{keygen}, \textsf{sid})$ from some party S, verify that $\textsf{sid} = (S, \textsf{sid}')$ for some $\textsf{sid}'$. If not, then ignore this request. Else, hand $(\texttt{keygen}, \textsf{sid})$ to the adversary. Upon receiving $(\texttt{VerificationKey}, \textsf{sid}, v)$ from the adversary, output $(\texttt{VerificationKey}, \textsf{sid}, v)$ to the caller S, and record the pair (S, v).

Sign. Upon receiving $(\texttt{sign}, \textsf{sid}, m)$ from S, verify that $\textsf{sid} = (S, \textsf{sid}')$ for some $\textsf{sid}'$. If not, then ignore this request. Else, send $(\texttt{sign}, \textsf{sid}, m)$ to the adversary. Upon receiving

($\texttt{signature}, \mathsf{sid}, m, \sigma$) from the adversary, verify that no entry $(m, \sigma, v, 0)$ is recorded. If it is, then output an error message to S and halt. Else, output ($\texttt{signature}, \mathsf{sid}, m, \sigma$) to S, and record the entry $(m, \sigma, v, 1)$.

Verify. On receiving the value ($\texttt{verify}, \mathsf{sid}, m, \sigma, v'$) from some party P, hand ($\texttt{verify}, \mathsf{sid}, m, \sigma, v'$) to the adversary. Upon receiving ($\texttt{verified}, \mathsf{sid}, m, \phi$) from the adversary, do:

1. If $v' = v$ and the entry $(m, \sigma, v, 1)$ is recorded, then set $b = 1$.
(This condition guarantees completeness: If the verification key v' is the registered one and σ is a legitimately generated signature for m, then the verification succeeds)
2. Else, if $v' = v$, the signer is not corrupted, and no entry $(m, \sigma', v, 1)$ for any σ' is recorded, then set $b = 0$ and record the entry $(m, \sigma, v, 0)$.
(This condition guarantees unforgeability: If v' is the registered one, the signer is not corrupted, and never signed m, then the verification fails.)
3. Else, if there is an entry (m, σ, v', b') recorded, then set $b = b'$.
(This condition guarantees consistency: All verification requests with identical parameters will result in the same answer.)
4. Else, let $b = \phi$ and record the entry (m, σ, v', ϕ).

Return ($\texttt{verified}, \mathsf{sid}, m, b$) to P.

A.2 The Second Generation Signature Functionality of Canetti [Can05]

Functionality A.2. $\mathcal{F}_{\mathsf{sig\text{-}2nd}}$ (Example Second-Generation Signature Functionality)

Key Generation. Upon receiving ($\texttt{keygen}, \mathsf{sid}$) from some party S, verify that $\mathsf{sid} = (S, \mathsf{sid}')$ for some sid'. If not, then ignore this request. Else, hand ($\texttt{keygen}, \mathsf{sid}$) to the adversary. Upon receiving ($\texttt{algs}, \mathsf{sid}, s, v$) from the adversary, where s is a description of a PPT ITM, and v is a description of a *deterministic* polytime ITM, output ($\texttt{VerificationAlgorithm}, \mathsf{sid}, v$) to S.

Sign. Upon receiving ($\texttt{sign}, \mathsf{sid}, m$) from S, let $\sigma = s(m)$, and verify that $v(m, \sigma) = 1$. If so, then output ($\texttt{signature}, \mathsf{sid}, m, \sigma$) to the caller P_i and record the entry (m, σ). Else, output an error message to S and halt.

Verify. On receiving the value ($\texttt{verify}, \mathsf{sid}, m, \sigma, v'$) from some party V do: If $v' = v$, the signer is not corrupted, $v(m, \sigma) = 1$, and no entry (m, σ') for any σ' is recorded, then output an error message to S and halt. Else, output ($\texttt{verified}, \mathsf{sid}, m, v'(m, \sigma)$) to V.

B Synchronous Protocols in UC (Continued)

In this section, we give material that complements Section 4.1, and in particular we include a high-level overview of the framework for universally composable synchronous computation from Katz et al. [KMTZ13]. The text that follows is taken almost verbatim from Cohen et al. [CCGZ17]. For the sake of self containment, we describe the basics of the model and introduce some terminology that simplifies the descriptions of corresponding functionalities.

Synchronous protocols can be cast as UC protocols that have access to a special clock functionality $\mathcal{F}_{\mathsf{clock}}$, which allows them to coordinate round switches as described below, and communicate over bounded-delay channels.[54] In a nutshell, the clock functionality works as follows: It stores a bit b which is initially set to 0, and it accepts from each party two types of messages: `clock-update` and `clock-read`. The functionality responds to `clock-read` by sending the value of b to the requestor. Each `clock-update` is forwarded to the adversary and also recorded, and when *all* honest parties have transmitted a `clock-update` message, the clock functionality updates b to $b \oplus 1$. It then continues as above, until it once again receives `clock-update` messages from all honest parties, at which point it resets b to $b \oplus 1$, and so on.

Such a clock can be used as follows to ensure that honest parties remain synchronized, i.e., no honest party proceeds to the next round before all (honest) parties have finished the current round: Every party stores a local variable where it keeps (its view of) the current value of the clock indicator b. At the beginning of the protocol execution this variable is 0 for all parties. In every round, every party uses all its activations (i.e., messages it receives) to complete all its current-round instructions, and only then sends `clock-update` to the clock signaling to the clock that it has completed its round. Following `clock-update`, all future activations result to the party sending `clock-read` to the clock until its bit b is flipped. Once the party observes that the bit b has flipped, it starts its next round. For the sake of clarity and brevity, we do not explicitly mention $\mathcal{F}_{\mathsf{clock}}$ in our constructions.

Katz et al. [KMTZ13] specify that for each message that is to be sent in the protocol, the sender and the receiver are given access to an independent single-use channel.[55] In this work we assume very simple CSFs that take as input from the sender the message it wishes to send (and a default input from other parties) and deliver the output to the receiver upon request. Such a simple secure-channel SFE can be realized in a straightforward manner from bounded-delay channels and a clock $\mathcal{F}_{\mathsf{clock}}$.

As is common in the synchronous protocols literature, throughout this work we will assume that protocols have the following structure: In each round every party sends/receives a (potentially empty) message to all parties and functionalities. Such protocols can be described in UC in a regular form using the methodology from Katz et al. [KMTZ13] as follows: Let $\mu \in \mathbb{N}$ denote the maximum number of messages that any party P_i might send to all recipients during some round.[56] Every party in the protocol uses exactly μ activations per round to complete its instructions. Specifically, when party P observes that the indicator-bit b of the clock has changed, P begins evaluating its instructions for the current round as described above. After each activation, it transmits one message, and after μ activations, it has transmitted all the messages that it needs to. Note that even if P does not need to transmit μ messages for some reason, it still waits for μ activations before progressing. Once μ activations have been received in the current round, P sends `clock-update` to the clock and thereafter it repeatedly sends `clock-read` messages every time it is activated, as

[54] As argued in Katz et al. [KMTZ13], bounded-delay channels are essential because they allow parties to detect whether or not a message was sent within a round.

[55] As pointed out by Katz et al., an alternative approach would be to have a multi-use communication channel. Modeling the actual communication network is out of scope for the current work, so we will use the more standard and formally treated model of single-use channels.

[56] In the simple case where the parties only use point-to-point channels, $\mu = 2(n-1)$, since each party uses $n-1$ channels as sender and $n-1$ as receiver to exchange its messages for each round with the $n-1$ other parties. Note that activating a channel as a *receiver* also requires sending a message to that channel's functionality, although this is usually implicit.

described above, until it observes b to change, whereafter the process repeats for the next round.

Katz et al. [KMTZ13] also described a way of capturing in UC the property that a protocol is guaranteed to terminate in a given number of rounds. They propose that if a synchronous protocol is expressed as described above and terminates after ρ_{term} rounds, then it realizes the functionality $\mathcal{F}$, which tracks the number of times every honest party activates it and delivers output to that honest party only after $\mu \cdot \rho_{\mathsf{term}}$ activations are received. More specifically, $\mathcal{F}$ imitates an ρ_{term}-round synchronous protocol with μ activations per party per round: upon being instantiated, $\mathcal{F}$ initializes a global round-counter $\tau := 0$ and an indicator variable $\tau_i := 0$ for each participating party P_i. Once P_i has activated $\mathcal{F}$ μ times,[57] $\mathcal{F}$ sets $\tau_i := 1$. If at this point $\tau_i = 1$ for *all* honest parties then $\mathcal{F}$ increments τ and resets $\tau_i = 0$ for every P_i. When $\tau = \rho_{\mathsf{term}}$, $\mathcal{F}$ enters a "delivery" mode, in which P_i can retrieve its output by transmitting `fetch-output` to $\mathcal{F}$.

We refer to a functionality that has the above structure, i.e., one that tracks of the current round τ by counting how many times every honest party sends μ of messages, as a *synchronous functionality.* To simplify the description of our functionalities, we introduce the following terminology. We say that *a synchronous functionality $\mathcal{F}$ is in round ρ* if the current value of the above internal counter in $\mathcal{F}$ is $\tau = \rho$.

We note that protocols in the synchronous model of Katz et al. [KMTZ13] enjoy the strong composition properties of the UC framework. However, in order to ensure that composed protocols are executed in lock-step, i.e., to ensure their round transitions are synchronized to the same clock ticks, Katz et al. use of the composition theorem for protocols with joint-state (JUC) [CR03]. In short, the parties run an $\mathcal{F}_{\mathsf{clock}}$-hybrid protocol $\hat{\pi}$ that emulates toward each of the other protocols the parties are concurrently running a sub-clock with a unique sub-session ID (ssid). Each sub-clock is local to its calling protocol, and $\hat{\pi}$ sends a `clock-update` signal to the actual (joint) clock functionality $\mathcal{F}_{\mathsf{clock}}$, only when all sub-clocks have received such a `clock-update` message. This ensures that all sub-clocks switch their internal bits at the same time, and the protocols using them are thus mutually synchronized. This property can be proven formally via direct application of the JUC theorem. For further details, we refer the reader to Katz et al. [KMTZ13] and Canetti and Rabin [CR03].

[57]To ensure that the simulator can keep track of the round index, $\mathcal{F}$ notifies the adversary about each received input, unless it has reached its delivery state defined below.